Introduction

This reference is for application programmers interested in creating OS/2 multimedia applications. It is also for subsystem developers who are interested in writing and installing subsystems to support specific data or devices. The IBM Developer's Toolkit for OS/2 Warp includes the bindings, header files, and libraries for development of OS/2 multimedia applications. OS/2 multimedia was referred to as Multimedia Presentation Manager/2 or MMPM/2 in previous releases.

Software Motion Video

The interface definitions for the digital video recording device are also provided in this reference. The digital video device uses *software-only* compression algorithms (*software motion video*) to enable playing or recording video without any additional video compression or decompression hardware.

Header Files

OS/2 multimedia includes header files with naming conventions compatible with the standard OS/2 format. Applications using previous versions of the MMPM/2 header files will still use those header files by default when the applications are compiled. In order to use the OS/2-consistent header files in an application, define INCL_OS2MM in the program before including the OS/2 multimedia system header file OS2ME.H. Defining INCL_OS2MM automatically defines the following:

INCL_MCIOS2	MCI-related include files (MCIOS2.H and MMDRVOS2.H)
INCL_MMIOOS2	MMIO include file (MMIOOS2.H)

The following additional header files have naming conventions compatible with the standard OS/2 format:

MIDIOS2.H
CDAUDOS2.H

Additional Multimedia Information

Multimedia REXX - (online)
Describes REXX functions that enable media control interface string commands to be sent from an OS/2 command file to control multimedia devices. This online book is provided with OS/2 multimedia.

Guide to Multimedia User Interface Design - (41G2922)
Describes design concepts to be considered when designing a CUA multimedia interface that is consistent within a particular multimedia product and across other products.

Using This Online Book

Before you begin to use this online book, it would be helpful to understand how you can:

- Expand the Contents to see all available topics
- Obtain additional information for a highlighted word or phrase
- Use action bar choices.

How To Use the Contents

When the Contents window first appears, some topics have a plus (+) sign beside them. The plus sign indicates that additional topics are available.

To expand the Contents if you are using a mouse, select the plus sign (+). If you are using a keyboard, use the Up or Down Arrow key to highlight the topic, and press the plus key (+).

To view a topic, double-click on the topic (or press the Up or Down Arrow key to highlight the topic, and then press Enter).

How To Obtain Additional Information

After you select a topic, the information for that topic appears in a window. Highlighted words or phrases indicate that additional information is available. You will notice that certain words in the following paragraph are highlighted in green letters, or in white letters on a black background. These are called hypertext terms. If you are using a mouse, double-click on the highlighted word. If you are using a keyboard, press the Tab key to move to the highlighted word, and then press the Enter key. Additional information will appear in a window.

How To Use Action Bar Choices

Several choices are available for managing information presented in the M-Control Program/2 Programming Reference. There are three pull-down menus on the action bar: the **Services** menu, the **Options** menu, and the **Help** menu.

The actions that are selectable from the **Services** menu operate on the active window currently displayed on the screen. These actions include the following:

Bookmark

Sets a place holder so you can retrieve information of interest to you.

When you place a bookmark on a topic, it is added to a list of bookmarks you have previously set. You can view the list, and you can remove one or all bookmarks from the list. If you have not set any bookmarks, the list is empty.

To set a bookmark, do the following:

1. Select a topic from the Contents.
2. When that topic appears, choose the **Bookmark** option from the **Services** menu.
3. If you want to change the name used for the bookmark, type the new name in the field.
4. Select the **Place** radio button (or press the Up or Down Arrow key to select it).
5. Select **OK**. The bookmark is then added to the bookmark list.

Search

Finds occurrences of a word or phrase in the current topic, selected topics, or all topics.

You can specify a word or phrase to be searched. You can also limit the search to a set of topics by first marking the topics in the Contents list.

To search for a word or phrase in all topics, do the following:

1. Choose the **Search** option from the **Services** pull-down.
2. Type the word or words to be searched.
3. Select **All sections**.
4. Select **Search** to begin the search.
5. The list of topics where the word or phrase appears is displayed.

Print

Prints one or more topics. You can also print a set of topics by first marking the topics in the Contents list.

You can print one or more topics. You can also print a set of topics by first marking the topics on the Contents list.

To print the document Contents list, do the following:

1. Select **Print** from the **Services** menu.
2. Select **Contents**.
3. Select **Print**.
4. The Contents list is printed on your printer.

Copy

Copies a topic you are viewing to a file you can edit.

You can copy a topic you are viewing into a temporary file named TEXT.TMP. You can later edit that file by using an editor such as the System Editor.

To copy a topic, do the following:

1. Expand the Contents list and select a topic.

2. When the topic appears, select **Copy to file** from the **Services** menu.

The system copies the text pertaining to that topic into the temporary TEXT.TMP file.

For information on any of the other choices in the **Services** menu, highlight the choice and press the F1 key.

Options
Changes the way the Contents is displayed.

You can control the appearance of the Contents list.

To expand the Contents and show all levels for all topics, select **Expand all** from the **Options** menu.

For information on any of the other choices in the **Options** menu, highlight the choice and press the F1 key.

What's New...

This release of the *OS/2 Multimedia Programming Reference* includes the following:

- Additional playlist commands:

  - SEMPOST_OPERATION
  - SEMWAIT_OPERATION

  See Memory Playlist Commands for a description of these commands.

- MCI_BUFFER and MCI_MIXSETUP messages and associated data structures. These messages enable use of the Direct Audio RouTines (DART), which allow applications to use a high-speed method of communication with the audio device.

- MCI_DOS_QUEUE flag for the MCI_OPEN message

- Enhanced DIVE capabilities including:

  - Transparent blitting to the screen using DiveSetTransparentBlitMode
  - Rotation of the output image when blitting to the screen (see the description of *fInvert* in the SETUP_BLITTER structure)
  - Blitting of changed lines using DiveBlitImageLines

- JPEGOPTIONS supporting extended JPEG I/O procedure information

- Additional flags for MCI_CUE to allow digital video devices to seek to a specified position and to display or hide the video window when cueing the media

- Real-Time MIDI Functions

- Reorganization of String Commands

  The string commands are organized into the following categories: System Commands, Required Commands, Basic Commands, and device-type specific command categories including CD audio, CD/XA, digital video, MIDI, videodisc player, video overlay, and waveform audio. Read the introduction to each of these sections carefully to understand how these categories relate to one another, where to find the string command you're looking for, and why it's located in the category that it is.

MCI Functions

The media control interface provides services to applications for controlling devices in the multimedia environment. These services are available through either a procedural message interface (mciSendCommand) or an interpretive string interface (mciSendString).

The following additional services are available to an application:

- Sharing devices with other applications
- Grouping devices for synchronization, acquisition, and collective use.

The media control interface uses the following functions for sending messages to control multimedia devices.

Function	Description
mciGetDeviceID	Retrieves the device ID corresponding to the alias of a device.
mciGetErrorString	Fills the caller's buffer with the error code string.
mciQuerySysValue	Queries OS/2 multimedia system values.
mciSendCommand	Sends a command to a media control driver using flags and structures.
mciSendString	Sends a command to a media device driver using string buffers.
mciSetSysValue	Sets or alters system wide values such as the captioning flag or working path for temporary files.

Note: To use the 16-bit versions of mciGetDeviceID, mciSendString, and mciGetErrorString, define INCL_16 in the source file using these functions. The 16-bit entry points provide 16-bit applications with the ability to use multimedia in the OS/2 environment. For example:

```
#define  INCL_MCIOS2
#define  INCL_16
#include <os2me.h>
```

----

mciGetDeviceID

----

mciGetDeviceID - Syntax

This function retrieves the device ID corresponding to an alias of a device. The ID can then be used on subsequent media control interface procedural commands. It also contains a 16-bit entry point.

```
#define INCL_MCIOS2
#include <os2.h>

PSZ      pszName;  /*  Alias name. */
ULONG    rc;       /*  Return code. */

rc = mciGetDeviceID(pszName);
```

----

mciGetDeviceID Parameter - pszName

pszName (PSZ) - input
    The alias name used with the open or connection command.

----

mciGetDeviceID Return Value - rc

rc (ULONG) - returns
    Returns the device ID assigned to this alias when the device was opened or when the connection command with the query flag was issued. Returns 0 if the alias name is not known or is invalid.

----

mciGetDeviceID - Parameters

pszName (PSZ) - input
    The alias name used with the open or connection command.

rc (ULONG) - returns
    Returns the device ID assigned to this alias when the device was opened or when the connection command with the query flag was issued. Returns 0 if the alias name is not known or is invalid.

----

mciGetDeviceID - Example Code

The following example illustrates how to retrieve a device ID.

```
 CHAR szBuffer[128];               /* Buffer for the string command */
 USHORT usDeviceID;                /* Return device ID              */

 strcpy(szBuffer,"open bell.wav alias wav1 wait");
                                         /* String command to open  */
                                         /* a wav file              */

 mciSendString ((PSZ)szBuffer,    /* Open a wav file                */
                NULL,             /* No return message              */
                0,                /* No return message length       */
                0,                /* No handle to callback          */
                0);               /* No notify parameter            */


usDeviceID = mciGetDeviceID((PSZ) "wav1");
                                  /* Returns device ID              */
                                  /* Assigned on the alias "wav1"   */
```

----

mciGetDeviceID - Topics

Select an item:

Syntax
Parameters
Returns
Example Code
Glossary

----

mciGetErrorString

----

mciGetErrorString - Syntax

This function fills the caller's buffer with the textual string associated with the given error code returned by the OS/2 multimedia function. It also contains a 16-bit entry point.

```
#define INCL_MCIOS2
#include <os2.h>

ULONG     ulError;    /*  Error code. */
PSZ       pszBuffer;  /*  Pointer to application's buffer. */
USHORT    usLength;   /*  Length of buffer. */
ULONG     rc;         /*  Return code. */

rc = mciGetErrorString(ulError, pszBuffer,
       usLength);
```

----

mciGetErrorString Parameter - ulError

ulError (ULONG) - input
Specifies the error code. The low-order word contains the error code and the high-order word contains the device ID. The device ID is used by OS/2 multimedia to determine if there are device-dependent errors. If there are device-dependent errors then OS/2 multimedia returns the device-dependent error string.

----

mciGetErrorString Parameter - pszBuffer

pszBuffer (PSZ) - output
Pointer to the application's buffer. The textual error string will be copied to this buffer based on the length of the buffer.

----

mciGetErrorString Parameter - usLength

usLength (USHORT) - input
Specifies the size of the application's buffer.

----

mciGetErrorString Return Value - rc

rc (ULONG) - returns
Return code.

MCIERR_SUCCESS
Error code returned indicating success or type of failure.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_OUTOFRANGE
The error code specified is not valid.

MCIERR_INVALID_BUFFER
The buffer address specified is not valid.

----

mciGetErrorString - Parameters

ulError (ULONG) - input
Specifies the error code. The low-order word contains the error code and the high-order word contains the device ID. The device ID is used by OS/2 multimedia to determine if there are device-dependent errors. If there are device-dependent errors then OS/2 multimedia returns the device-dependent error string.

pszBuffer (PSZ) - output
Pointer to the application's buffer. The textual error string will be copied to this buffer based on the length of the buffer.

usLength (USHORT) - input
Specifies the size of the application's buffer.

rc (ULONG) - returns
Return code.

MCIERR_SUCCESS
Error code returned indicating success or type of failure.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_OUTOFRANGE
The error code specified is not valid.

MCIERR_INVALID_BUFFER
The buffer address specified is not valid.

----

mciGetErrorString - Remarks

The maximum string length returned is 128 bytes. If the size of the application's buffer (*usLength*) is smaller than the size of the error string to be returned, then only *usLength* bytes of the error string will be copied into the application's buffer. Therefore, a buffer size of 128 bytes is recommended to avoid this problem.

----

mciGetErrorString - Example Code

The following code illustrates how to obtain the description of a given error code.

```
#define ILLEGAL_COMMAND  (USHORT) 0x0000FFFF /* Illegal command   */
#define MCI_ERROR_STRING_LENGTH  128         /* Length of error
                                                message buffer    */

CHAR acErrorStringBuffer[MCI_ERROR_STRING_LENGTH];
ULONG ulRC;

ulRC =
  mciSendCommand(
     0,                            /* Don't know the device yet     */
     ILLEGAL_COMMAND,              /* Command to be performed       */
     MCI_WAIT,                     /* Flags for the command         */
     (ULONG) NULL,                 /* No parameter list             */
     0 );                          /* No notify message             */

if ( ulRC != MCIERR_SUCCESS)
 {
  ulRC =
   mciGetErrorString(
     ulRC,
     (PSZ) acErrorStringBuffer,          /* acErrorStringBuffer     */
     (USHORT) MCI_ERROR_STRING_LENGTH );/* should = "unrecognized
                                           command"                 */
 }
```

----

mciGetErrorString - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Example Code
Glossary

----

mciQuerySysValue

----

mciQuerySysValue - Syntax

This function queries the value of system-defined attributes.

```
#define INCL_MCIOS2
#include <os2.h>

USHORT    iSysValue;  /*  System attribute. */
PVOID     pValue;     /*  Pointer to return field. */
BOOL      rc;         /*  Return code. */

rc = mciQuerySysValue(iSysValue, pValue);
```

----

mciQuerySysValue Parameter - iSysValue

iSysValue (USHORT) - input
Specifies the system attribute. The possible system attributes are:

MSV_CLOSEDCAPTION
Returns TRUE if the user has enabled closed captioning and FALSE otherwise.

MSV_MASTERVOLUME
The master volume setting. The range is 0 to 100.

MSV_HEADPHONES
Returns TRUE if the user has headphones enabled for the system and FALSE otherwise.

MSV_SPEAKERS
Returns TRUE if the user has speakers (line out) enabled for the system and FALSE otherwise.

MSV_WORKPATH
Points to a character buffer of size CCHMAXPATH. This is the name of the file-system path where temporary files created by OS/2 multimedia are located (for example, c:\mmos2\temp).

MSV_SYSQOSVALUE
System wide Quality of Service (QOS) specification value used for band-width reservation (for example, bytes per second) over the network.

MSV_SYSQOSERRORFLAG
Description of error occurring during band-width reservation.

----

mciQuerySysValue Parameter - pValue

pValue (PVOID) - in/out
Pointer to the return field. The type of data object this field points to is dependent on the attribute requested:

System Attribute	Data Type
MSV_CLOSEDCAPTION	BOOL
MSV_MASTERVOLUME	ULONG
MSV_HEADPHONES	ULONG
MSV_SPEAKERS	ULONG
MSV_WORKPATH	PSZ
MSV_SYSQOSVALUE	ULONG
MSV_SYSQOSERRORFLAG	ULONG

----

mciQuerySysValue Return Value - rc

rc (BOOL) - returns
If the command completes successfully then MCIERR_SUCCESS is returned, otherwise non-zero is returned.

----

mciQuerySysValue - Parameters

iSysValue (USHORT) - input
Specifies the system attribute. The possible system attributes are:

MSV_CLOSEDCAPTION
Returns TRUE if the user has enabled closed captioning and FALSE otherwise.

MSV_MASTERVOLUME
The master volume setting. The range is 0 to 100.

MSV_HEADPHONES
Returns TRUE if the user has headphones enabled for the system and FALSE otherwise.

MSV_SPEAKERS
Returns TRUE if the user has speakers (line out) enabled for the system and FALSE otherwise.

MSV_WORKPATH
Points to a character buffer of size CCHMAXPATH. This is the name of the file-system path where temporary files created by OS/2 multimedia are located (for example, c:\mmos2\temp).

MSV_SYSQOSVALUE
System wide Quality of Service (QOS) specification value used for band-width reservation (for example, bytes per second) over the network.

MSV_SYSQOSERRORFLAG
Description of error occurring during band-width reservation.

pValue (PVOID) - in/out
Pointer to the return field. The type of data object this field points to is dependent on the attribute requested:

System Attribute	Data Type
MSV_CLOSEDCAPTION	BOOL
MSV_MASTERVOLUME	ULONG
MSV_HEADPHONES	ULONG
MSV_SPEAKERS	ULONG
MSV_WORKPATH	PSZ
MSV_SYSQOSVALUE	ULONG
MSV_SYSQOSERRORFLAG	ULONG

rc (BOOL) - returns
If the command completes successfully then MCIERR_SUCCESS is returned, otherwise non-zero is returned.

----

mciQuerySysValue - Related Functions

- mciSetSysValue

----

mciQuerySysValue - Example Code

The following code illustrates how to query a multimedia system value.

```
  #define INCL_MCIOS2
  #include <os2me.h>

  CHAR  szWorkPath[CCHMAXPATH];
  mciQuerySysValue(MSV_WORKPATH,  szWorkPath); /* Get temporary
                                                    file path.    */
```

----

mciQuerySysValue - Topics

Select an item:
Syntax
Parameters
Returns
Example Code
Related Functions
Glossary

----

mciSendCommand

----

mciSendCommand - Syntax

This function sends a media control interface message to the specified media device.

```
#define INCL_MCIOS2
#include <os2.h>

USHORT    usDeviceID;  /*  Device ID. */
USHORT    usMessage;   /*  Message action. */
ULONG     ulParam1;    /*  Message flags. */
PVOID     pParam2;     /*  Message data. */
USHORT    usUserParm;  /*  User-specified parameter. */
ULONG     rc;          /*  Return code. */
```

```
rc = mciSendCommand(usDeviceID, usMessage,
       ulParam1, pParam2, usUserParm);
```

----

mciSendCommand Parameter - usDeviceID

usDeviceID (USHORT) - input
The device ID the message is to be sent to. This is the device ID returned from MCI_OPEN; this parameter is ignored on the MCI_OPEN message.

----

mciSendCommand Parameter - usMessage

usMessage (USHORT) - input
The media control interface message to send. See MCI Command Messages for descriptions of these messages.

----

mciSendCommand Parameter - ulParam1

ulParam1 (ULONG) - input
Flags for this message. These flags are defined separately for each message; however, the following flags are available for *all* media control interface messages unless denoted in the message description. MCI_NOTIFY and MCI_WAIT are mutually exclusive.

MCI_NOTIFY
A notification message (MM_MCINOTIFY) will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not returned until the action indicated by this message is completed or an error occurs.

----

mciSendCommand Parameter - pParam2

pParam2 (PVOID) - input
Pointer to a data structure for this message. These structures are defined separately for each message.

----

mciSendCommand Parameter - usUserParm

usUserParm (USHORT) - input

User parameter returned in the notification for this message.

----

mciSendCommand Return Value - rc

rc (ULONG) - returns
Returns MCIERR_SUCCESS in the low-order word if there was no error; otherwise it returns the error code in the low-order word of the return value.

Use mciGetErrorString to convert this code to a textual string. If the return code is a device-dependent error, the high-order word will contain the device ID. See Return Codes for a listing of possible return values. If the MCI_NOTIFY flag is specified then the device receiving this message performs error checking to see if it can begin processing the message. The amount of required error checking varies depending on the message and device. The device returns to the application and the rest of the command processing occurs asynchronously.

----

mciSendCommand - Parameters

usDeviceID (USHORT) - input
The device ID the message is to be sent to. This is the device ID returned from MCI_OPEN; this parameter is ignored on the MCI_OPEN message.

usMessage (USHORT) - input
The media control interface message to send. See MCI Command Messages for descriptions of these messages.

ulParam1 (ULONG) - input
Flags for this message. These flags are defined separately for each message; however, the following flags are available for *all* media control interface messages unless denoted in the message description. MCI_NOTIFY and MCI_WAIT are mutually exclusive.

MCI_NOTIFY
A notification message (MM_MCINOTIFY) will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not returned until the action indicated by this message is completed or an error occurs.

pParam2 (PVOID) - input
Pointer to a data structure for this message. These structures are defined separately for each message.

usUserParm (USHORT) - input
User parameter returned in the notification for this message.

rc (ULONG) - returns
Returns MCIERR_SUCCESS in the low-order word if there was no error; otherwise it returns the error code in the low-order word of the return value.

Use mciGetErrorString to convert this code to a textual string. If the return code is a device-dependent error, the high-order word will contain the device ID. See Return Codes for a listing of possible return values. If the MCI_NOTIFY flag is specified then the device receiving this message performs error checking to see if it can begin processing the message. The amount of required error checking varies depending on the message and device. The device returns to the application and the rest of the command processing occurs asynchronously.

----

mciSendCommand - Remarks

Use mciSendString to send textual command strings. The mciSendString function calls an internal string parser to parse the string and sends the resulting structure to mciSendCommand.

----

mciSendCommand - Related Functions

- mciGetDeviceID
- mciGetErrorString
- mciSendString

----

mciSendCommand - Example Code

The following code illustrates how to send a command to a specified device.

```
MCI_OPEN_PARMS mciOpenParameters;
MCI_PLAY_PARMS mciPlayParameters;
CHAR DeviceType[] = "cdaudio";
                                /* Device type "cdaudio"          */

mciPlayParameters.hwndCallback = PM_Win_Handle;
                                /* Assign hwndCallback the handle
                                   to the PM Window routine       */

mciOpenParameters.pszDeviceType = (PSZ)&DeviceType;


mciSendCommand(
 0,                                /* Don't know the device yet   */
 MCI_OPEN,                         /* MCI message                 */
 MCI_WAIT | MCI_OPEN_TYPE_ID,      /* Flags for the MCI
                                      message                     */
 (PVOID) &mciOpenParameters,       /* Parameters for the message  */
 0 );                              /* No notify message           */



mciSendCommand(
   mciOpenParameters.usDeviceID,   /* Device to play the cdaudio  */
   MCI_PLAY,                        /* MCI message                */
   MCI_WAIT,                        /* Flags for the MCI message  */
   (PVOID) &mciPlayParameters,      /* Parameters for the message */
   0);                              /* No notify message          */


 mciSendCommand(
  mciOpenParameters.usDeviceID,     /* Device to play the cdaudio */
  MCI_CLOSE,                        /* MCI message                */
  MCI_WAIT,                         /* Flags for the MCI message  */
  (PVOID) NULL,                     /* No Parameter list          */
  0);                               /* No notify message          */
```

----

mciSendCommand - Topics

Select an item:

---

mciSendString

---

mciSendString - Syntax

This function sends a media control interface command string to a media device. It also contains a 16-bit entry point.

```
#define INCL_MCIOS2
#include <os2.h>

PSZ       pszCommandBuf;    /*  Media control command string. */
PSZ       pszReturnString;  /*  Application-supplied buffer. */
USHORT    usReturnLength;   /*  Bytes reserved. */
HWND      hwndCallBack;     /*  Window handle. */
USHORT    usUserParm;       /*  User-specified parameter. */
ULONG     rc;               /*  Return code. */

rc = mciSendString(pszCommandBuf, pszReturnString,
       usReturnLength, hwndCallBack, usUserParm);
```

---

mciSendString Parameter - pszCommandBuf

pszCommandBuf (PSZ) - input
Media control command string of the form:

```
<command> <object> <keywords>
```

The object can be the device type, file name, alias, and so forth.

---

mciSendString Parameter - pszReturnString

pszReturnString (PSZ) - output
A application-supplied buffer for the return data. This pointer can be NULL if no return information is desired. For more information see String Commands.

----

mciSendString Parameter - usReturnLength

usReturnLength (USHORT) - input
The number of bytes reserved for *pszReturnString* .

----

mciSendString Parameter - hwndCallBack

hwndCallBack (HWND) - input
A PM window handle to be used in returning asynchronous notification messages. This parameter must be specified if **notify** was specified in the command string.

----

mciSendString Parameter - usUserParm

usUserParm (USHORT) - input
User parameter returned in the notification for this message.

----

mciSendString Return Value - rc

rc (ULONG) - returns
Returns MCIERR_SUCCESS in the low-order word if there was no error; otherwise it returns an error code in the low-order word of the return value. Use mciGetErrorString to convert this code to a string. If the error code is a device-dependent error, the high-order word will contain the device ID.

----

mciSendString - Parameters

pszCommandBuf (PSZ) - input
Media control command string of the form:

```
<command> <object> <keywords>
```

The object can be the device type, file name, alias, and so forth.

pszReturnString (PSZ) - output
A application-supplied buffer for the return data. This pointer can be NULL if no return information is desired. For more information see String Commands.

usReturnLength (USHORT) - input
: The number of bytes reserved for *pszReturnString* .

hwndCallBack (HWND) - input
: A PM window handle to be used in returning asynchronous notification messages. This parameter must be specified if **notify** was specified in the command string.

usUserParm (USHORT) - input
: User parameter returned in the notification for this message.

rc (ULONG) - returns
: Returns MCIERR_SUCCESS in the low-order word if there was no error; otherwise it returns an error code in the low-order word of the return value. Use mciGetErrorString to convert this code to a string. If the error code is a device-dependent error, the high-order word will contain the device ID.

----

mciSendString - Remarks

If *pszReturnString* is NULL or *usReturnLength* is 0, no data will be returned.

If the return code is MCIERR_SUCCESS and the command does return data (such as status), the string parser will convert the return data to string format if appropriate. An example is **status cdaudio media present** would return TRUE or FALSE. If the application requests the return value to be converted to a string by the string parser, it must specify the WAIT flag. See String Commands for a description of the media control interface strings and return values.

----

mciSendString - Related Functions

- mciGetDeviceID
- mciGetErrorString
- mciSendCommand

----

mciSendString - Example Code

The following code illustrates how to send a command to a specified device.

```
CHAR szBuffer[128];                 /* String command buffer     */

strcpy (szBuffer, "open bell.wav alias wav1 wait");
                                    /* String command to open     */

mciSendString ((PSZ)szBuffer,       /* Open a wav file            */
               NULL,                /* No return data             */
               0,                   /* No return length           */
               0,                   /* No window callback handle  */
               0);                  /* No notify message          */

strcpy (szBuffer, "play wav1 wait");/* String command to play     */

mciSendString ((PSZ)szBuffer,       /* Play a wav file            */
               NULL,                /* No return data             */
               0,                   /* No return length           */
               0,                   /* No window callback handle  */
               0);                  /* No notify message          */

strcpy (szBuffer, "close wav1 wait");/* String command to close   */
```

```
     mciSendString ((PSZ)szBuffer,       /* Close a wav file           */
                    NULL,                /* No return data             */
                    0,                   /* No return length           */
                    0,                   /* No window callback handle  */
                    0);                  /* No notify message          */
```

---

mciSendString - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Example Code
Related Functions
Glossary

---

mciSetSysValue

---

mciSetSysValue - Syntax

This function sets the value of system-defined attributes.

```
#include <os2.h>

USHORT    iSysValue;  /*  System attribute. */
PVOID     pValue;     /*  Pointer to value. */
BOOL      rc;         /*  Return code. */

rc = mciSetSysValue(iSysValue, pValue);
```

---

mciSetSysValue Parameter - iSysValue

iSysValue (USHORT) - input
     The system attribute. See mciQuerySysValue for a list of possible system attributes.

---

mciSetSysValue Parameter - pValue

pValue (PVOID) - input
Pointer to a value to be set. The type of data object this points to is dependent on the attribute requested. See mciQuerySysValue for a list of data types.

----

mciSetSysValue Return Value - rc

rc (BOOL) - returns
Return code.

TRUE	If the function succeeds.
FALSE	If the function fails.

----

mciSetSysValue - Parameters

iSysValue (USHORT) - input
The system attribute. See mciQuerySysValue for a list of possible system attributes.

pValue (PVOID) - input
Pointer to a value to be set. The type of data object this points to is dependent on the attribute requested. See mciQuerySysValue for a list of data types.

rc (BOOL) - returns
Return code.

TRUE	If the function succeeds.
FALSE	If the function fails.

----

mciSetSysValue - Remarks

Most of the system values can be changed by way of the Multimedia Setup program to reflect the preferences of the end user. In general, other applications should only query these values.

----

mciSetSysValue - Related Functions

- mciQuerySysValue

----

mciSetSysValue - Example Code

The following code illustrates how to set a multimedia system value.

```
  /* Turn closed captioning flag on so
     applications will provide captioning */

     mciSetSysValue  (MSV_CLOSEDCAPTION, TRUE);
```

----

mciSetSysValue - Topics

Select an item:

High-Level Macro Service Functions

The high-level macro service functions provide general playback and recording within a single function. These functions hide the programming overhead associated with playing and recording multimedia data, such as opening and closing a device, and simplify using multimedia capabilities in applications.

Note: mciPlayFile and mciPlayResource play different types of data (audio, video, MIDI, and so forth), however mciRecordAudioFile records *only* digital audio.

The high-level functions are listed in the following table.

Function	Description
mciPlayFile	Plays a multimedia file or audio elements of a compound file.
mciPlayResource	Plays a multimedia resource that has been bound into an application.
mciRecordAudioFile	Records digital audio into a file specified by the caller. Records only digital audio.

To use the 16-bit versions of mciPlayFile, mciPlayResource, and mciRecordAudioFile, define INCL_16 in the source file using these functions. The 16-bit entry points provide 16-bit applications with the ability to use multimedia in the OS/2 environment. For example:

```
#define  INCL_MACHDR
#define  INCL_16
#include <os2me.h>
```

----

mciPlayFile

----

mciPlayFile - Syntax

This function plays a multimedia data file, (such as digital audio or video), or a digital audio element of a RIFF compound file, using media control interface commands. It opens, plays, and closes the file. mciPlayFile is a 32-bit function that is also provided as a 16-bit entry point.

The mciPlayFile function requires a message queue.

```
#define INCL_MACHDR
#define INCL_MCIOS2
#include <os2.h>

HWND     hwndOwner;     /*  Window handle. */
PSZ      pszFile;       /*  Pointer to file name. */
ULONG    ulFlags;       /*  Flags. */
PSZ      pszTitle;      /*  Window title. */
HWND     hwndViewport;  /*  Window handle for video image. */
ULONG    rc;            /*  Return code. */

rc = mciPlayFile(hwndOwner, pszFile, ulFlags,
       pszTitle, hwndViewport);
```

----

mciPlayFile Parameter - hwndOwner

hwndOwner (HWND) - input
    Window handle of the owner window. If this parameter is NULL, the currently active window is used.

----

mciPlayFile Parameter - pszFile

pszFile (PSZ) - input
    Pointer to a multimedia file name. Compound-file names are also supported. For example:

```
a:\path\file+element
```

----

mciPlayFile Parameter - ulFlags

ulFlags (ULONG) - input

MCI_OWNERISPARENT
: Indicates that the owner window should be used as the parent window for any default window that is created. If this flag is passed to a device that does not support a parent window, an error is returned.

MCI_STOPACTIVE
: Indicates that any currently active command issued by either mciPlayFile or mciPlayResource should be stopped.

MCI_ASYNC
: Indicates that the command should be processed asynchronously. A rendezvous command will not be done.

MCI_ASYNCRENDEZVOUS
: Indicates that the command should proceed asynchronously. A rendezvous command will be done.

MCI_RENDEZVOUS
: Indicates that the call should wait for a currently pending asynchronous command to complete.

- If no command is pending, then it returns immediately.
- If an asynchronous command is not pending, this function will return immediately. This flag indicates that the command should wait until a pending asynchronous play command completes and then return.
- If a synchronous (default) play command is pending, this command should return immediately with an MCIERR_NO_ASYNC_PLAY_ACTIVE.
- If another MCI_RENDEZVOUS command is pending, this command should return immediately with an MCIERR_NO_ASYNC_PLAY_ACTIVE.

----

mciPlayFile Parameter - pszTitle

pszTitle (PSZ) - input
: Title for window if one is generated. The title is ignored if a window would not be generated. (For example, an audio file is to be played).

----

mciPlayFile Parameter - hwndViewport

hwndViewport (HWND) - input
: Window handle for displaying the video image. If a viewport window is not specified, then a default video window is displayed. This parameter only has an effect when the data type supports video.

----

mciPlayFile Return Value - rc

rc (ULONG) - returns
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_NO_ASYNC_PLAY_ACTIVE
A synchronous (default) play command is pending or no asynchronous play is currently active for the associated owner window.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_FILE_ATTRIBUTE
File is read only, or is opened for write mode by other application.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Can be returned if another application has opened or acquired the device for exclusive use. Issue MCI_ACQUIREDEVICE to activate the device ID.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags can not be used together.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_DUPLICATE_ALIAS
Alias already exists.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_CANNOT_LOAD_DRIVER
The driver could not be loaded.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_OUT_OF_MEMORY
Out of memory.

----

mciPlayFile - Parameters

hwndOwner (HWND) - input
Window handle of the owner window. If this parameter is NULL, the currently active window is used.

pszFile (PSZ) - input
Pointer to a multimedia file name. Compound-file names are also supported. For example:

```
a:\path\file+element
```

ulFlags (ULONG) - input

MCI_OWNERISPARENT
Indicates that the owner window should be used as the parent window for any default window that is created. If this flag is passed to a device that does not support a parent window, an error is returned.

MCI_STOPACTIVE
: Indicates that any currently active command issued by either mciPlayFile or mciPlayResource should be stopped.

MCI_ASYNC
: Indicates that the command should be processed asynchronously. A rendezvous command will not be done.

MCI_ASYNCRENDEZVOUS
: Indicates that the command should proceed asynchronously. A rendezvous command will be done.

MCI_RENDEZVOUS
: Indicates that the call should wait for a currently pending asynchronous command to complete.

- If no command is pending, then it returns immediately.
- If an asynchronous command is not pending, this function will return immediately. This flag indicates that the command should wait until a pending asynchronous play command completes and then return.
- If a synchronous (default) play command is pending, this command should return immediately with an MCIERR_NO_ASYNC_PLAY_ACTIVE.
- If another MCI_RENDEZVOUS command is pending, this command should return immediately with an MCIERR_NO_ASYNC_PLAY_ACTIVE.

pszTitle (PSZ) - input
Title for window if one is generated. The title is ignored if a window would not be generated. (For example, an audio file is to be played).

hwndViewport (HWND) - input
Window handle for displaying the video image. If a viewport window is not specified, then a default video window is displayed. This parameter only has an effect when the data type supports video.

rc (ULONG) - returns
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_NO_ASYNC_PLAY_ACTIVE
: A synchronous (default) play command is pending or no asynchronous play is currently active for the associated owner window.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCIERR_FILE_ATTRIBUTE
: File is read only, or is opened for write mode by other application.

MCIERR_INSTANCE_INACTIVE
: The device is currently inactive. Can be returned if another application has opened or acquired the device for exclusive use. Issue MCI_ACQUIREDEVICE to activate the device ID.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
: Unsupported function.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags can not be used together.

MCIERR_FILE_NOT_FOUND
: File has not been loaded.

MCIERR_DUPLICATE_ALIAS
: Alias already exists.

MCIERR_INVALID_BUFFER

Invalid return buffer given.

MCIERR_CANNOT_LOAD_DRIVER
The driver could not be loaded.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_OUT_OF_MEMORY
Out of memory.

----

mciPlayFile - Remarks

This function provides a simple way of playing a multimedia data file. It supports any multimedia file type or RIFF compound files.

The audio is played on the default media control interface device. A device control panel is not displayed for audio.

Still images are not supported.

For video, the default media control interface driver window is displayed. The movie is played from beginning to end. The window is destroyed when the device is closed. If an *hwndViewport* window is specified, then the video will be shown in the viewport window.

The default is to play the file synchronously unless the MCI_ASYNC or MCI_ASYNCRENDEZVOUS flag is specified. The message queue is processed during its processing.

When the file name that is passed is a NULL pointer or an empty buffer, then an MCIERR_MISSING_PARAMETER error is returned unless the MCI_STOPACTIVE or MCI_RENDEZVOUS flags are set. In order to stop a currently active command, use the MCI_STOPACTIVE flag.

Either mciPlayFile or mciPlayResource could return an MCIERR_NO_ASYNC_PLAY_ACTIVE error. This error indicates that no asynchronous play is currently active for the associated owner window.

The title parameter can be NULL. If a title is specified and a window is displayed, the title is used as the window title. A window is only displayed if a video file is played.

When the *pszFile* parameter is specified and there is an active PLAY command associated with the specified owner window, the first command is superceded by the second command.

----

mciPlayFile - Related Functions

- mciPlayResource
- mciRecordAudioFile
- mmioRemoveElement
- mmioFindElement

----

mciPlayFile - Example Code

The following code illustrates how to play a digital audio file.

```
#define INCL_MCIOS2
#define INCL_MACHDR
#include <os2me.h>         /* Play a wave file           */
                           /* set to valid window handle */
ULONG   rc;
HWND    hwnd;
```

```
rc=mciPlayFile ( hwnd, "GONG.WAV", 0,0,0);
```

---

mciPlayFile - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Example Code
Related Functions
Glossary

---

mciPlayResource

---

mciPlayResource - Syntax

This function plays a multimedia resource, such as a waveform, MIDI, or video, on the default device associated with the resource type. mciPlayResource is a 32-bit function that is also provided as a 16-bit entry point.

```
#define INCL_MACHDR
#define INCL_MCIOS2
#include <os2.h>

HWND       hwndOwner;     /*  Window handle. */
HMODULE    hmod;          /*  Module handle. */
ULONG      resType;       /*  Resource type. */
ULONG      resID;         /*  Resource identifier. */
ULONG      ulFlags;       /*  Flags. */
PSZ        pszTitle;      /*  Window title. */
HWND       hwndViewport;  /*  Window handle. */
ULONG      rc;            /*  Return code. */

rc = mciPlayResource(hwndOwner, hmod, resType,
       resID, ulFlags, pszTitle, hwndViewport);
```

---

mciPlayResource Parameter - hwndOwner

hwndOwner (HWND) - input
Window handle of the owner window. If this parameter is NULL then the currently active window is used.

---

mciPlayResource Parameter - hmod

hmod (HMODULE) - input
Module handle of the module that contains the resource. The resource is loaded using DosGetResource. NULL indicates the program file's resources.

----

mciPlayResource Parameter - resType

resType (ULONG) - input
Defines resource type with one of the following values:

RT_WAVE	Resource type is digital audio.
RT_AVI	Resource type is digital video using the AVI file format.
RT_RMID	Resource type is MIDI.
RT_RIFF	Resource type is RIFF. Any of the resource types can be contained within this resource type.

----

mciPlayResource Parameter - resID

resID (ULONG) - input
Identifier for resource.

----

mciPlayResource Parameter - ulFlags

ulFlags (ULONG) - input

MCI_OWNERISPARENT	Indicates that the owner window should be used as the parent window for any default window that is created. If this flag is passed to a device that does not support a parent window, an error is returned.
MCI_STOPACTIVE	Indicates that any currently active PLAY command issued by mciPlayFile or mciPlayResource should be stopped.
MCI_ASYNC	Indicates that the command should be processed asynchronously. A rendezvous command will not be done.
MCI_ASYNCRENDEZVOUS	Indicates that the command should be processed asynchronously. A rendezvous command will be done.

MCI_RENDEZVOUS
Indicates that the call should wait for a currently pending asynchronous command to complete.

- If no command is pending, then it returns immediately.
- If an asynchronous command is not pending, this function returns immediately. This flag indicates that the command should wait until a pending asynchronous play command completes and then return.
- If a synchronous (default) play command is pending, this command returns immediately with a MCIERR_NO_ASYNC_PLAY_ACTIVE.
- If another MCI_RENDEZVOUS command is pending, this command should return immediately with a MCIERR_ASYNC_PLAY_ACTIVE.

----

mciPlayResource Parameter - pszTitle

pszTitle (PSZ) - input
Title for window if one is generated. The title is ignored if a window would not be generated.

----

mciPlayResource Parameter - hwndViewport

hwndViewport (HWND) - input
Window handle for displaying the video image. If a viewport window is not specified, then a default video window is displayed. This parameter only has an effect when the data type supports video.

----

mciPlayResource Return Value - rc

rc (ULONG) - returns
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_NO_ASYNC_PLAY_ACTIVE
A synchronous (default) play command is pending or no asynchronous play is currently active for the associated owner window.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_FILE_ATTRIBUTE
Returned if another application has opened or acquired the same device for exclusive use.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE message to activate device ID.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE

Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags can not be used together.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_DUPLICATE_ALIAS
Alias already exists.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_CANNOT_LOAD_DRIVER
The driver could not be loaded.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_OUT_OF_MEMORY
Out of memory.

----

mciPlayResource - Parameters

hwndOwner (HWND) - input
Window handle of the owner window. If this parameter is NULL then the currently active window is used.

hmod (HMODULE) - input
Module handle of the module that contains the resource. The resource is loaded using DosGetResource. NULL indicates the program file's resources.

resType (ULONG) - input
Defines resource type with one of the following values:

RT_WAVE
Resource type is digital audio.

RT_AVI
Resource type is digital video using the AVI file format.

RT_RMID
Resource type is MIDI.

RT_RIFF
Resource type is RIFF. Any of the resource types can be contained within this resource type.

resID (ULONG) - input
Identifier for resource.

ulFlags (ULONG) - input

MCI_OWNERISPARENT
Indicates that the owner window should be used as the parent window for any default window that is created. If this flag is passed to a device that does not support a parent window, an error is returned.

MCI_STOPACTIVE
Indicates that any currently active PLAY command issued by mciPlayFile or mciPlayResource should be stopped.

MCI_ASYNC

Indicates that the command should be processed asynchronously. A rendezvous command will not be done.

MCI_ASYNCRENDEZVOUS
: Indicates that the command should be processed asynchronously. A rendezvous command will be done.

MCI_RENDEZVOUS
: Indicates that the call should wait for a currently pending asynchronous command to complete.

- If no command is pending, then it returns immediately.
- If an asynchronous command is not pending, this function returns immediately. This flag indicates that the command should wait until a pending asynchronous play command completes and then return.
- If a synchronous (default) play command is pending, this command returns immediately with a MCIERR_NO_ASYNC_PLAY_ACTIVE.
- If another MCI_RENDEZVOUS command is pending, this command should return immediately with a MCIERR_ASYNC_PLAY_ACTIVE.

pszTitle (PSZ) - input
: Title for window if one is generated. The title is ignored if a window would not be generated.

hwndViewport (HWND) - input
: Window handle for displaying the video image. If a viewport window is not specified, then a default video window is displayed. This parameter only has an effect when the data type supports video.

rc (ULONG) - returns
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_NO_ASYNC_PLAY_ACTIVE
: A synchronous (default) play command is pending or no asynchronous play is currently active for the associated owner window.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCIERR_FILE_ATTRIBUTE
: Returned if another application has opened or acquired the same device for exclusive use.

MCIERR_INSTANCE_INACTIVE
: The device is currently inactive. Issue MCI_ACQUIREDEVICE message to activate device ID.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
: Unsupported function.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags can not be used together.

MCIERR_FILE_NOT_FOUND
: File has not been loaded.

MCIERR_DUPLICATE_ALIAS
: Alias already exists.

MCIERR_INVALID_BUFFER
: Invalid return buffer given.

MCIERR_CANNOT_LOAD_DRIVER
: The driver could not be loaded.

MCIERR_DEVICE_LOCKED
: The device is acquired for exclusive use.

MCIERR_OUT_OF_MEMORY
Out of memory.

----

mciPlayResource - Remarks

This function provides a simple way of playing a multimedia resource stored in a program resource.

The audio is played on the default media control interface device. A device control panel is not displayed for audio.

Still images are not supported.

For video, the default media control interface driver window is displayed. The movie is played from beginning to end. The window is destroyed when the device is closed. If an *hwndViewport* window is specified, then the video will be shown in the viewport window.

The default is to play the resource synchronously unless the MCI_ASYNC or MCI_ASYNCRENDEZVOUS flag is specified. The message queue is processed during its processing.

Either mciPlayFile or mciPlayResource could return an MCIERR_NO_ASYNC_PLAY_ACTIVE error. This error indicates that no asynchronous play is currently active for the associated owner window.

The title parameter can be NULL. If a title is specified and a window is displayed, the title is used as the window title. A window is only displayed if a video file is played.

If the *resID* is 0, MCIERR_MISSING_PARAMETER is returned unless the MCI_STOPACTIVE or MCI_RENDEZVOUS flags are set. To stop a currently active command, use the MCI_STOPACTIVE flag.

----

mciPlayResource - Related Functions

- mciPlayFile
- mciRecordAudioFile
- mmioRemoveElement
- mmioFindElement

----

mciPlayResource - Example Code

Bring the appropriate multimedia files into your resource file as shown below:

```
#include <os2medef.h>
RESOURCE RT_WAVE IDR_WAVE "zipper.wav"
RESOURCE RT_RMID IDR_MIDI "bach.mid"
RESOURCE RT_AVI  IDR_ULT "\\mmos2\\movies\\macaw.avi"
RESOURCE RT_RIFF IDR_WAVE "zipper.wav"
RESOURCE RT_RIFF IDR_MIDI "bach.mid"
RESOURCE RT_RIFF IDR_ULT "macaw.avi"
```

The RT_* values are the "resource types" and the IDR_* values are the resource identifiers you provide. Refer to the *PM Programming Guide and Reference* for detailed information on creating resource files.

You can then use mciPlayResource to play a multimedia resource through the media control interface as shown below:

```
#define INCL_MACHDR
#define INCL_MCIOS2
#include <os2me.h>
rc = mciPlayResource( hwnd,       /* Window handle */
                      hmod,       /* Resource module handle or 0 for EXE */
                      RT_WAVE,    /* Resource type */
                      IDR_WAVE,   /* Resource ID */
                      ulFlags,
                      szTitle,    /* Other API values */
                      hwndClient);
```

---

mciPlayResource - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Example Code
Related Functions
Glossary

---

mciRecordAudioFile

---

mciRecordAudioFile - Syntax

This function records an audio file or MMIO compound audio file element. mciRecordAudioFile is a 32-bit function that is also provided as a 16-bit entry point.

The mciRecordAudioFile function requires a message queue and focus window.

```
#define INCL_MACHDR
#define INCL_MCIOS2
#include <os2.h>

HWND     hwndOwner;  /*  Window handle. */
PSZ      pszFile;    /*  Pointer to file name. */
PSZ      pszTitle;   /*  Recorder window title. */
ULONG    ulFlags;    /*  Reserved. */
ULONG    rc;         /*  Return code. */

rc = mciRecordAudioFile(hwndOwner, pszFile,
       pszTitle, ulFlags);
```

---

mciRecordAudioFile Parameter - hwndOwner

hwndOwner (HWND) - input
The window handle of the owner window. If this parameter is NULL then the currently active window is used.

----

mciRecordAudioFile Parameter - pszFile

pszFile (PSZ) - input
Pointer to a multimedia file name. Compound-file names are also supported. For example:

```
a:\path\file+element
```

----

mciRecordAudioFile Parameter - pszTitle

pszTitle (PSZ) - input
Specifies the title for the recorder window.

----

mciRecordAudioFile Parameter - ulFlags

ulFlags (ULONG) - input
Reserved for future use and must be set to zero.

----

mciRecordAudioFile Return Value - rc

rc (ULONG) - returns
Returns MCIERR_SUCCESS if there was no error. An escape from the recorder dialog returns the DID_CANCEL return code.

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_UNSUPPORTED_FLAG
ulFlags is not set to zero.

MCIERR_MISSING_PARAMETER
No file name is sent.

MCIERR_FILE_NOT_FOUND
The filename is a NULL string.

MCIERR_OUT_OF_MEMORY
MMPM/2 could not allocate memory.

DID_CANCEL
User cancelled from recording without saving recorded files, or there was an MCI error.

----

mciRecordAudioFile - Parameters

hwndOwner (HWND) - input
The window handle of the owner window. If this parameter is NULL then the currently active window is used.

pszFile (PSZ) - input
Pointer to a multimedia file name. Compound-file names are also supported. For example:

```
a:\path\file+element
```

pszTitle (PSZ) - input
Specifies the title for the recorder window.

ulFlags (ULONG) - input
Reserved for future use and must be set to zero.

rc (ULONG) - returns
Returns MCIERR_SUCCESS if there was no error. An escape from the recorder dialog returns the DID_CANCEL return code.

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_UNSUPPORTED_FLAG
ulFlags is not set to zero.

MCIERR_MISSING_PARAMETER
No file name is sent.

MCIERR_FILE_NOT_FOUND
The filename is a NULL string.

MCIERR_OUT_OF_MEMORY
MMPM/2 could not allocate memory.

DID_CANCEL
User cancelled from recording without saving recorded files, or there was an MCI error.

----

mciRecordAudioFile - Remarks

The mciRecordAudioFile function provides a small, simple recorder window, which allows an object-oriented method of recording audio annotations. All play and record operations are from beginning to end.

This call does not return until the recorder window is closed. The message queue is processed during the operation of this function. Once the recording is completed, the window is dismissed.

This function records 11 kHz, mono, PCM audio data from the microphone input of the default waveaudio device. The sample size defaults to the card default.

This function creates the file if it doesn't exist. If a compound-file name is specified (d:\path\file+element), the file will be created. If it doesn't exist, the element will be created after the record operation completes.

The *pszFile* parameter, which specifies the name of the object to record into, is an input-only parameter.

When *pszTitle* is not specified, the last component of the file name or the MMIO element name is used.

This function records *only* digital audio files.

----

mciRecordAudioFile - Related Functions

- mciPlayFile
- mciPlayResource
- mmioRemoveElement
- mmioFindElement

----

mciRecordAudioFile - Example Code

The following code illustrates how to record an audio file.

```
#define INCL_MCIOS2
#define INCL_MACHDR
#include <os2me.h>

ULONG rc;
HWND  hwnd;
rc=mciRecordAudioFile (hwnd, "SOUND.WAV", "TITLE", 0);
```

----

mciRecordAudioFile - Topics

Select an item:
Syntax
Parameters
Returns
Remarks
Example Code
Related Functions
Glossary

Subsystem Messages

The MCIDRV commands provide subsystem communication between MDM and the MCDs. The current set of MCIDRV commands provide for device resource management. The MCIDRV_SAVE and MCIDRV_RESTORE messages allow MDM to manage devices that support multiple device contexts either concurrently or serially. The MCIDRV_CHANGERESOURCE message allows MCDs to change the resource consumed by a device context as required. MCIDRV_CHANGERESOURCE is sent from an MCD to MDM. A device context is made active when the MCD receives an MCIDRV_RESTORE from MDM. An MCI_OPEN command is not complete (the device is not active) until MDM has sent the MCD an MCIDRV_RESTORE. Similarly, when MDM sends an MCD the MCIDRV_SAVE command, the MCD will make the device context inactive. These commands provide multiple device contexts the ability to share one device.

Message	**Description**
MCIDRV_CHANGERESOURCE	Changes the class or resource units assigned to the given device context.
MCIDRV_RESTORE	Restores state of an inactive device context.
MCIDRV_SAVE	Saves state of a device context.

----

mdmDriverNotify

----

mdmDriverNotify - Syntax

This function is called from MCDs to return message to applications. Returned information includes command status, cuepoints, position changes, playlist messages, and device specific events.

```
#define INCL_MMIO
#include <os2.h>

USHORT    usDeviceID;  /*  Device ID for message. */
HWND      hwnd;        /*  Window handle. */
USHORT    usMsgType;   /*  Notification type. */
USHORT    usUserParm;  /*  User-defined. */
ULONG     ulMsgParm;   /*  Message-defined. */
ULONG     rc;          /*  Return codes. */

rc = mdmDriverNotify(usDeviceID, hwnd, usMsgType,
       usUserParm, ulMsgParm);
```

----

mdmDriverNotify Parameter - usDeviceID

usDeviceID (USHORT) - input
Device ID to be assoicated with this message.

----

mdmDriverNotify Parameter - hwnd

hwnd (HWND) - input
The window handle used to post or send message to application.

----

mdmDriverNotify Parameter - usMsgType

usMsgType (USHORT) - input
Type of notification:

- MM_MCICUEPOINT
- MM_MCIEVENT
- MM_MCINOTIFY
- MM_MCIPASSDEVICE
- MM_MCIPLAYLISTMESSAGE
- MM_MCIPOSITIONCHANGE

---

mdmDriverNotify Parameter - usUserParm

usUserParm (USHORT) - input
User-defined parameter.

---

mdmDriverNotify Parameter - ulMsgParm

ulMsgParm (ULONG) - input
Message-defined parameter.

---

mdmDriverNotify Return Value - rc

rc (ULONG) - returns
Return codes indicating success or type of failure:

MCI_NOTIFY_SUCCESS	If the function succeeds, 0 is returned.
MM_MCIPOSITIONCHANGE	The media position in MMTIME units.
MM_MCICUEPOINT	Media position in MMTIME units.
MM_MCIPLAYLISTMESSAGE	Parameter specified by playlist message instruction (Operand 2).
MM_MCIEVENT	Device-specific parameter.

---

mdmDriverNotify - Parameters

usDeviceID (USHORT) - input
Device ID to be assoicated with this message.

hwnd (HWND) - input
The window handle used to post or send message to application.

usMsgType (USHORT) - input
Type of notification:

- MM_MCICUEPOINT
- MM_MCIEVENT
- MM_MCINOTIFY
- MM_MCIPASSDEVICE
- MM_MCIPLAYLISTMESSAGE
- MM_MCIPOSITIONCHANGE

usUserParm (USHORT) - input
User-defined parameter.

ulMsgParm (ULONG) - input
Message-defined parameter.

rc (ULONG) - returns
Return codes indicating success or type of failure:

MCI_NOTIFY_SUCCESS	If the function succeeds, 0 is returned.
MM_MCIPOSITIONCHANGE	The media position in MMTIME units.
MM_MCICUEPOINT	Media position in MMTIME units.
MM_MCIPLAYLISTMESSAGE	Parameter specified by playlist message instruction (Operand 2).
MM_MCIEVENT	Device-specific parameter.

----

mdmDriverNotify - Topics

Select an item:
Syntax
Parameters
Returns
Glossary

MCIDRV_CHANGERESOURCE

MCIDRV_CHANGERESOURCE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following standard flag:

MCI_WAIT	This message is not to be returned until the device context resource requirements have been changed or an error is found.

MCIDRV_CHANGERESOURCE Parameter - pParam2

pParam2 (PMCIDRV_CHANGERESOURCE_PARMS)
A pointer to the MCIDRV_CHANGERESOURCE_PARMS structure.

MCIDRV_CHANGERESOURCE - Description

This message is sent from the MCDs to MDM to change the class and resource units assigned to the given device context.

ulParam1 (ULONG)
This parameter can contain the following standard flag:

MCI_WAIT	This message is not to be returned until the device context resource requirements have been changed or an error is found.

pParam2 (PMCIDRV_CHANGERESOURCE_PARMS)
A pointer to the MCIDRV_CHANGERESOURCE_PARMS structure.

MCIDRV_CHANGERESOURCE - Topics

Select an item:
Description
Glossary

MCIDRV_RESTORE

MCIDRV_RESTORE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following standard flags:

MCI_WAIT	The message is not to be returned until the device context restore is complete.
MCI_SHAREABLE	Device context is in shareable mode.
MCI_EXCLUSIVE	Device context is in exclusive mode.

MCIDRV_RESTORE Parameter - pParam2

pParam2 (PVOID)
Not used.

MCIDRV_RESTORE - Description

This message is sent from MDM to MCDs to restore the state of an inactive device context. If this message is received for an active device context then the MCD should save the shareability for the device instance. This is either shareable or exclusive. See *ulParam1* for more details.

ulParam1 (ULONG)
This parameter can contain the following standard flags:

MCI_WAIT	The message is not to be returned until the device context restore is complete.
MCI_SHAREABLE	Device context is in shareable mode.
MCI_EXCLUSIVE	Device context is in exclusive mode.

pParam2 (PVOID)
Not used.

MCIDRV_RESTORE - Topics

Select an item:
Description
Glossary

MCIDRV_SAVE

MCIDRV_SAVE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following standard flags:

MCI_WAIT	Control is not to be returned until the action indicated by this message is completed.

MCIDRV_SAVE Parameter - pParam2

pParam2 (PVOID)
Not used.

MCIDRV_SAVE - Description

This message is sent from MDM to MCDs to save the state of an active device context.

ulParam1 (ULONG)
This parameter can contain the following standard flags:

MCI_WAIT	Control is not to be returned until the action indicated by this message is completed.

pParam2 (PVOID)
Not used.

MCIDRV_SAVE - Topics

Select an item:
Description
Glossary

Notification Messages

The system uses notification messages to respond to applications, indicating system status such as completion of a media device function or passing of the ownership of a media device between processes.

Messages are returned to applications asynchronously (using WinPostMsg), except for MM_MCIEVENT, which is sent synchronously (using WinSendMsg). A media control interface call that results in the dispatch of these two messages (such as MCI_OPEN and MCI_ACQUIREDEVICE) must be issued from application threads that have a message queue.

All messages except system messages operate in an asynchronous mode without notification unless MCI_NOTIFY or MCI_WAIT is specified. These two flags are mutually exclusive. If both are used, an MCIERR_FLAGS_NOT_COMPATIBLE error is returned. If MCI_WAIT is used, control is not returned to the caller until the command completes. MCI_NOTIFY returns control to the caller and then completes the command. A notification will be sent to the application if MCIERR_SUCCESS was returned on the call. The second parameter specified for each message is a pointer to a control block structure associated with that message. This pointer is passed in the *pParam2* parameter of mciSendCommand.

Function	Description
MM_MCICUEPOINT	Notifies application that a cue point is found in a playlist, or that a cue point has been detected, which was set with the MCI_SET_CUEPOINT message.
MM_MCIEVENT	Notifies application of an event generated by a device.
MM_MCINOTIFY	Notifies an application after a device completes action or an error occurs.
MM_MCIPASSDEVICE	Notifies application that a shared device is being gained or lost.
MM_MCIPLAYLISTMESSAGE	Notifies application that playlist processor has found a MESSAGE instruction.
MM_MCIPOSITIONCHANGE	Notifies applications of current media position.

MM_MCICUEPOINT

MM_MCICUEPOINT Field - usUserParameter

usUserParameter (USHORT)
User parameter specified in the MCI_CUEPOINT_PARMS structure when the cue point was set.

MM_MCICUEPOINT Field - usDeviceID

usDeviceID (USHORT)
   Device ID.

MM_MCICUEPOINT Field - ulMMtime

ulMMtime (ULONG)
   Media position in MMTIME units.

MM_MCICUEPOINT - Parameters

usUserParameter (USHORT)
   User parameter specified in the MCI_CUEPOINT_PARMS structure when the cue point was set.

usDeviceID (USHORT)
   Device ID.

ulMMtime (ULONG)
   Media position in MMTIME units.

MM_MCICUEPOINT - Description

This message notifies an application that the device has encountered a cue point in a playlist, or that a cue point has been set with MCI_SET_CUEPOINT.

```
MsgParam1
     USHORT  usUserParameter  /*  User-specified parameter. */
     USHORT  usDeviceID       /*  Device ID. */

MsgParam2
     ULONG   ulMMtime         /*  Media position. */
```

MM_MCICUEPOINT - Remarks

MM_MCICUEPOINT is returned to the window procedure that sent the MCI_SET_CUEPOINT message.

MM_MCICUEPOINT - Topics

Select an item:
Description
Parameters
Remarks
Glossary

MM_MCIEVENT

MM_MCIEVENT Field - usEventCode

usEventCode (USHORT)
Device-specific event code. The following event notification codes are currently defined:

MCI_MIXEVENT	A mixer attribute has changed.

MM_MCIEVENT Field - usDeviceID

usDeviceID (USHORT)
Device ID.

MM_MCIEVENT Field - pEventData

pEventData (PVOID)
Device-specific event data structure.

MM_MCIEVENT Return Value - ulReserved

ulReserved (ULONG)
Zero. Reserved value.

MM_MCIEVENT - Parameters

usEventCode (USHORT)
  Device-specific event code. The following event notification codes are currently defined:

  MCI_MIXEVENT
    A mixer attribute has changed.

usDeviceID (USHORT)
  Device ID.

pEventData (PVOID)
  Device-specific event data structure.

ulReserved (ULONG)
  Zero. Reserved value.

MM_MCIEVENT - Description

This message notifies an application of an event generated by a device.

```
MsgParam1
     USHORT  usEventCode  /*  Device-specific event code. */
     USHORT  usDeviceID   /*  Device ID. */

MsgParam2
     PVOID   pEventData   /*  Device-specific event data. */
```

MM_MCIEVENT - Remarks

The format of the data structure pointed to by *MsgParam2* is defined by devices that return this message.

Unlike most media control interface notification messages, MM_MCIEVENT is sent (rather than posted) to the application's message queue. The data structure pointed to by the message parameter is considered valid only during processing of the message.

MM_MCIEVENT - Topics

Select an item:
Description
Parameters
Returns
Remarks
Glossary

MM_MCINOTIFY

MM_MCINOTIFY Field - usNotifycode

usNotifycode (USHORT)
Specifies the following notification message code:

MCI_NOTIFY_SUCCESSFUL	The command was completed successfully.
MCI_NOTIFY_SUPERSEDED	Another notification request (same type of command) was received.
MCI_NOTIFY_ABORTED	The command was interrupted and is unable to be completed. For example, the first command was a PLAY with notify, and the second command was STOP with or without notify.

Any other value indicates an error, and that value is the error number. mciGetErrorString can be used to convert the number into a textual description of the error.

MM_MCINOTIFY Field - usUserParameter

usUserParameter (USHORT)
Specifies a *usUserParameter* notification message code.

Contains the user parameter specified on mciSendCommand or mciSendString for this command.

MM_MCINOTIFY Field - usDeviceID

usDeviceID (USHORT)
The media control interface device ID included in the notification.

MM_MCINOTIFY Field - usMessage

usMessage (USHORT)
Specifies the message ID which generated the notification.

MM_MCINOTIFY - Parameters

usNotifycode (USHORT)
Specifies the following notification message code:

MCI_NOTIFY_SUCCESSFUL
The command was completed successfully.

MCI_NOTIFY_SUPERSEDED
Another notification request (same type of command) was received.

MCI_NOTIFY_ABORTED
The command was interrupted and is unable to be completed. For example, the first command was a PLAY with notify, and the second command was STOP with or without notify.

Any other value indicates an error, and that value is the error number. mciGetErrorString can be used to convert the number into a textual description of the error.

usUserParameter (USHORT)
Specifies a *usUserParameter* notification message code.

Contains the user parameter specified on mciSendCommand or mciSendString for this command.

usDeviceID (USHORT)
The media control interface device ID included in the notification.

usMessage (USHORT)
Specifies the message ID which generated the notification.

MM_MCINOTIFY - Description

This message notifies an application when a device completes the action indicated by a media message or when an error occurs.

```
MsgParam1
     USHORT  usNotifycode     /*  Notification code. */
     USHORT  usUserParameter  /*  User parameter. */

MsgParam2
     USHORT  usDeviceID       /*  Device ID. */
     USHORT  usMessage        /*  Message ID. */
```

MM_MCINOTIFY - Topics

Select an item:
Description
Parameters
Glossary

MM_MCIPASSDEVICE

MM_MCIPASSDEVICE Field - usDeviceID

usDeviceID (USHORT)
    Device ID.

MM_MCIPASSDEVICE Field - usReserved

usReserved (USHORT)
    Reserved.

MM_MCIPASSDEVICE Field - usEvent

usEvent (USHORT)
    Indicates whether use of the device is being gained or lost (MCI_GAINING_USE or MCI_LOSING_USE).

MM_MCIPASSDEVICE Field - usReserved

usReserved (USHORT)
    Reserved.

MM_MCIPASSDEVICE - Parameters

usDeviceID (USHORT)
    Device ID.

usReserved (USHORT)
    Reserved.

usEvent (USHORT)
    Indicates whether use of the device is being gained or lost (MCI_GAINING_USE or MCI_LOSING_USE).

usReserved (USHORT)
Reserved.

MM_MCIPASSDEVICE - Description

This message notifies an application that the use of a device is being gained or lost.

```
MsgParam1
     USHORT  usDeviceID   /*  Device ID. */
     USHORT  usReserved   /*  Reserved. */

MsgParam2
     USHORT  usEvent      /*  Gaining or losing use of device. */
     USHORT  usReserved   /*  Reserved. */
```

MM_MCIPASSDEVICE - Remarks

The window handle specified in the *hwndCallback* field of the structure passed with the MCI_OPEN command is used as the window handle for the MM_MCIPASSDEVICE messages.

MM_MCIPASSDEVICE - Topics

Select an item:
Description
Parameters
Remarks
Glossary

MM_MCIPLAYLISTMESSAGE

MM_MCIPLAYLISTMESSAGE Field - usInstruction

usInstruction (USHORT)
Playlist instruction number.

MM_MCIPLAYLISTMESSAGE Field - usDeviceID

usDeviceID (USHORT)
Device ID.

MM_MCIPLAYLISTMESSAGE Field - ulMessageParm

ulMessageParm (ULONG)
Parameter specified in playlist MESSAGE instruction (operand 2).

MM_MCIPLAYLISTMESSAGE - Parameters

usInstruction (USHORT)
Playlist instruction number.

usDeviceID (USHORT)
Device ID.

ulMessageParm (ULONG)
Parameter specified in playlist MESSAGE instruction (operand 2).

MM_MCIPLAYLISTMESSAGE - Description

This message notifies an application that the playlist processor has encountered a MESSAGE instruction.

```
MsgParam1
     USHORT  usInstruction  /*  Playlist instruction number. */
     USHORT  usDeviceID     /*  Device ID. */

MsgParam2
     ULONG   ulMessageParm  /*  Playlist parameter. */
```

MM_MCIPLAYLISTMESSAGE - Topics

Select an item:
Description

Parameters
Glossary

MM_MCIPOSITIONCHANGE

MM_MCIPOSITIONCHANGE Field - usUserParameter

usUserParameter (USHORT)
User parameter specified in the MCI_POSITION_PARMS structure when position advise notification was requested.

MM_MCIPOSITIONCHANGE Field - usDeviceID

usDeviceID (USHORT)
Device ID.

MM_MCIPOSITIONCHANGE Field - ulMMtime

ulMMtime (ULONG)
Media position in MMTIME units.

MM_MCIPOSITIONCHANGE - Parameters

usUserParameter (USHORT)
User parameter specified in the MCI_POSITION_PARMS structure when position advise notification was requested.

usDeviceID (USHORT)
Device ID.

ulMMtime (ULONG)
Media position in MMTIME units.

MM_MCIPOSITIONCHANGE - Description

This message notifies an application of the current media position.

```
MsgParam1
     USHORT  usUserParameter   /*  User-specified parameter. */
     USHORT  usDeviceID        /*  Device ID. */

MsgParam2
     ULONG   ulMMtime          /*  Media position. */
```

MM_MCIPOSITIONCHANGE - Remarks

This message is generated only periodically during a recording or playback operation if the MCI_SET_POSITION_ADVISE message has been sent to the device to enable position advise notifications. This message is posted to the window handle that was specified on the MCI_SET_POSITION_ADVISE message.

MM_MCIPOSITIONCHANGE - Topics

Select an item:
Description
Parameters
Remarks
Glossary

MCI Command Messages

This section describes the media control interface command messages.

All messages except system messages operate in an asynchronous mode without notification unless MCI_NOTIFY or MCI_WAIT is specified. These two flags are mutually exclusive. If both are used, the error MCIERR_FLAGS_NOT_COMPATIBLE is returned.

If MCI_WAIT is used, control is not returned to the caller until the command completes. MCI_NOTIFY returns control to the caller and then completes the command. A notification will be sent to the application if MCIERR_SUCCESS was returned on the call. The second parameter specified for each message is a pointer to a control block structure associated with that message. This pointer is passed in the *pParam2* parameter of mciSendCommand. The following table lists the command messages.

Command	Description
MCI_ACQUIREDEVICE	Requests the use of the media device.
MCI_BUFFER	Allows an application to allocate (or deallocate) buffers for use with the audio device.
MCI_CAPTURE	Causes a video device to capture the current video image.
MCI_CLOSE	Closes a device.
MCI_CONNECTION	Queries the device ID of a

	connected device.
MCI_CONNECTOR	Enables or disables a connector, or to query the status of a connector.
MCI_CONNECTORINFO	Determines the total number of connectors on a device, the number of connectors of a specific type, the type of each of the connectors, and whether or not a particular type of connection is valid for a connector.
MCI_COPY	Copies data from the device element to the clipboard or a user-supplied buffer.
MCI_CUE	Signals a device to ready itself (preroll) so that a subsequent playback or recording operation begins with minimum delay.
MCI_CUT	Removes data from the device element and copies it to the clipboard or a user-supplied buffer.
MCI_DEFAULT_CONNECTION	Makes, breaks, and queries default connections between devices.
MCI_DELETE	Removes the specified range of data from the device element.
MCI_DEVICESETTINGS	Allows a media control driver (MCD) to insert custom settings pages into a Settings notebook.
MCI_ESCAPE	Sends a string directly to the driver.
MCI_FREEZE	Freezes the motion of a video image.
MCI_GETDEVCAPS	Returns static information about a particular driver.
MCI_GETIMAGEBUFFER	Reads data from the image capture buffer.
MCI_GETIMAGEPALETTE	Obtains a palette or color map for the current image.
MCI_GETTOC	Interrogates the device, and returns a table of contents structure for the currently loaded disk. (CD Audio Only)
MCI_GROUP	Used to provide the appropriate message handling for GROUP commands. GROUP commands allow you to control several multimedia devices from a single MCI command.
MCI_INFO	Returns string information from a media device.
MCI_LOAD	Specifies a new file or RIFF chunk to be loaded into an already existing device context.
MCI_MASTERAUDIO	Provides support for setting and retrieving system-wide audio control parameters.
MCI_MIXNOTIFY	Notifies an application of mixer attribute changes.
MCI_MIXSETUP	Informs the mixer device that the application wishes to read or write

	buffers directly and sets up the device in the correct mode.
MCI_OPEN	Opens a logical multimedia device and creates a new device context for use by an application.
MCI_PASTE	Pastes data from the clipboard or a user-supplied buffer into the specified range of a device element.
MCI_PAUSE	Suspends playback or recording.
MCI_PLAY	Signals the device to begin transmitting data.
MCI_PUT	Sets the source and destination rectangle arrays for the transformation of the video image.
MCI_RECORD	Starts the device recording input data.
MCI_REDO	Redoes the record, cut, paste, or delete operation most recently undone by MCI_UNDO.
MCI_RELEASEDEVICE	Releases the exclusive use of physical device resources by a device context or device group.
MCI_RESTORE	Causes a video device to restore the image or bitmap.
MCI_RESUME	Resumes playing or recording from a paused state.
MCI_REWIND	Seeks the media to the beginning point.
MCI_SAVE	This message saves the current file.
MCI_SEEK	Changes the current media position of the device.
MCI_SET	Sets device information.
MCI_SET_CUEPOINT	Sets run-time cue points in the media device.
MCI_SETIMAGEBUFFER	Writes data to the image capture buffer.
MCI_SETIMAGEPALETTE	Sets a palette or color map to be used for mapping images.
MCI_SET_POSITION_ADVISE	Enables periodic position-change messages from the media device.
MCI_SET_SYNC_OFFSET	Specifies positional offsets for devices operating in synchronization.
MCI_SETTUNER	Sets the frequency for the tuner device.
MCI_SPIN	Spins the player up or down.
MCI_STATUS	Obtains information about the status of a media control interface device.
MCI_STEP	Steps the player one or more frames.
MCI_STOP	Stops audio or video playback or recording.

MCI_SYSINFO	Returns information about media control interface devices.
MCI_UNDO	Undoes the operation most recently performed by record, cut, paste, or delete.
MCI_UNFREEZE	Restores motion to an area of the display frozen with MCI_FREEZE.
MCI_WHERE	Returns the extent of the clipping rectangles.
MCI_WINDOW	Specifies the window and the window characteristics that a graphic device should use for display.

MCI_ACQUIREDEVICE

MCI_ACQUIREDEVICE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ACQUIRE_QUEUE
An MCI_ACQUIREDEVICE message is queued and executed as soon as device resources are available. If the request can be satisfied immediately, then it is not queued. If an MCI_ACQUIREDEVICE message is queued and an MCI_RELEASEDEVICE or MCI_CLOSE message is sent for that instance, the queued MCI_ACQUIREDEVICE message is cancelled.

MCI_EXCLUSIVE
Resources are to be exclusively allocated for the device instance. Exclusive use of resources can be released with an MCI_RELEASEDEVICE message.

MCI_EXCLUSIVE_INSTANCE
Acquires the device instance for exclusive use without acquiring the entire device resource for exclusive use. This flag locks the device instance and prevents it from being made inactive until the application sends an MCI_RELEASEDEVICE or MCI_CLOSE message. The MCI_RELEASEDEVICE puts the instance back into the fully shareable state.

MCI_ACQUIREDEVICE Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_ACQUIREDEVICE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
The function is successful.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_CALLBACK_HANDLE
The callback handle given is not correct.

MCI_ACQUIREDEVICE - Description

This message requests that the given device instance be made active. It is also used to request either exclusive or exclusive instance rights for this instance.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ACQUIRE_QUEUE
An MCI_ACQUIREDEVICE message is queued and executed as soon as device resources are available. If the request can be satisfied immediately, then it is not queued. If an MCI_ACQUIREDEVICE message is queued and an MCI_RELEASEDEVICE or MCI_CLOSE message is sent for that instance, the queued MCI_ACQUIREDEVICE message is cancelled.

MCI_EXCLUSIVE
Resources are to be exclusively allocated for the device instance. Exclusive use of resources can be released with an MCI_RELEASEDEVICE message.

MCI_EXCLUSIVE_INSTANCE
Acquires the device instance for exclusive use without acquiring the entire device resource for exclusive use. This flag locks the device instance and prevents it from being made inactive until the application sends an

MCI_RELEASEDEVICE or MCI_CLOSE message. The MCI_RELEASEDEVICE puts the instance back into the fully shareable state.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	The function is successful.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_DEVICE_LOCKED	The device is acquired for exclusive use.
MCIERR_INVALID_FLAG	Flag is invalid (*ulParam1*).
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_INVALID_CALLBACK_HANDLE	The callback handle given is not correct.

MCI_ACQUIREDEVICE - Remarks

The application can specify *exclusive access* , which inhibits other applications from acquiring use of the device until released by the owning application.

When a device is opened by an application, the physical device resource is acquired automatically by the newly created device instance. If the device instance subsequently loses use of the physical resource, it can regain use later by issuing MCI_ACQUIREDEVICE. This message enables applications to participate in a device-sharing scheme, driven by WM_ACTIVATE message processing, wherein the use of physical devices generally is granted to the application with which the user is interacting by the application issuing MCI_ACQUIREDEVICE.

If a defined device instance loses use of the physical device to other device instances, that use is regained when the other device instances are closed, even if MCI_ACQUIREDEVICE is not issued.

When a process acquires use of a shared device that currently is in use by another process, the device instance is saved for the previous process.

Applications receive the MM_MCIPASSDEVICE message whenever they gain or lose use of a device. Use of a device is not obtained until the MM_MCIPASSDEVICE message is received. This message is posted (by way of WinPostMsg) to the window handle specified in the *hwndCallback* field on the MCI_OPEN message. If an invalid or no *hwndCallback* parameter is provided on the MCI_OPEN message, then no MM_MCIPASSDEVICE messages are received.

If the device has been acquired exclusively by another device instance, the function returns MCIERR_DEVICE_LOCKED.

MCI_ACQUIREDEVICE - Related Messages

- MCI_OPEN
- MCI_RELEASEDEVICE

MCI_ACQUIREDEVICE - Example Code

The following code illustrates how an application can acquire a device.

```
MCI_GENERIC_PARMS mciGenericParms;      /* Info data structure for cmd */
USHORT   usDeviceID;                    /* Device ID                   */
HWND     hwndMyWindow;                  /* Handle to the PM window     */
MPARAM   mp1;                           /* Message parameter passed    */
                                           on window procedure message */

            /* Assign hwndCallback the handle to the PM window routine */

  mciGenericParms.hwndCallback =  hwndMyWindow;


            /* Acquire the device if our window is being activated. */

  if ((BOOL)mp1)
    {
     mciSendCommand(usDeviceID,         /* Requested device           */
                  MCI_ACQUIREDEVICE,    /* MCI acquire device message */
                  MCI_NOTIFY,           /* Flags for this message     */
                  (PVOID)&mciGenericParms,
                                        /* Parameter data structure   */
                  0);                   /* No user parameter for      */
                                        /* notification message       */
    }
```

MCI_ACQUIREDEVICE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_BUFFER

MCI_BUFFER Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* field of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ALLOCATE_MEMORY
Allocates memory for the mixer.

MCI_DEALLOCATE_MEMORY
Deallocates memory from the mixer.

MCI_BUFFER Parameter - pParam2

pParam2 (PMCI_BUFFER_PARMS)
A pointer to an MCI_BUFFER_PARMS data structure.

MCI_BUFFER Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_INVALID_BUFFER
Buffer specified in the *pBufList* field of the MCI_BUFFER_PARMS structure is invalid.

MCIERR_INVALID_MODE
Command invalid for current mode.

MCIERR_OUT_OF_MEMORY
Memory could not be allocated.

MCI_BUFFER - Description

This message allows an application to allocate (or deallocate) buffers for use with the audio device. Buffers are limited to 64K on Intel machines.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* field of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ALLOCATE_MEMORY
: Allocates memory for the mixer.

MCI_DEALLOCATE_MEMORY
: Deallocates memory from the mixer.

pParam2 (PMCI_BUFFER_PARMS)
A pointer to an MCI_BUFFER_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_INVALID_BUFFER
: Buffer specified in the *pBufList* field of the MCI_BUFFER_PARMS structure is invalid.

MCIERR_INVALID_MODE
: Command invalid for current mode.

MCIERR_OUT_OF_MEMORY
: Memory could not be allocated.

MCI_BUFFER - Remarks

On input, MCI_BUFFER_PARMS should contain the number of buffers to be allocated, the size for each buffer, and a pointer to an array of MCI_MIX_BUFFER structures (one per buffer).

The mixer will attempt to allocate the number of buffers and size of buffers to use. If the mixer cannot satisfy the entire request, it will update the *ulNumBuffers* field with the total number of buffers that it was able to allocate. If no memory could be allocated, MCIERR_OUT_OF_MEMORY will be returned. If memory has already been allocated, and the MCI_ALLOCATE_BUFFER flag is used, MCIERR_INVALID_MODE is returned.

MCI_BUFFER - Related Messages

- MCI_MIXSETUP

MCI_BUFFER - Example Code

The following example illustrates using MCI_BUFFER to allocate memory.

```
 MCI_MIX_BUFFER  MyBuffers[ MAX_BUFFERS ];
```

```
BufferParms.ulNumBuffers = 40;
BufferParms.ulBufferSize = 4096;
BufferParms.pBufList = MyBuffers;

rc = mciSendCommand( usDeviceID,
                   MCI_BUFFER,
                   MCI_WAIT | MCI_ALLOCATE_MEMORY,
                   ( PVOID ) &BufferParms,
                   0 );

if ( ULONG_LOWD( rc ) != MCIERR_SUCCESS )
   {
   printf( "Error allocating memory.  rc is: %d", rc );
   exit ( 1 );
   }

/*  MCI driver will return the number of buffers     */
/*  it was able to allocate.                         */
/*  It will also return the size of the information  */
/*  allocated with each buffer.                      */

ulNumBuffers = BufferParms.ulNumBuffers;

for ( ulLoop = 0; ulLoop < ulNumBuffers; ulLoop++ )
   {
   rc = mmioRead ( hmmio,
                   MyBuffers[ ulLoop ].pBuffer,
                   MyBuffers[ ulLoop ].ulBufferLength );

   if ( !rc )
      {
      exit( rc );
      }
   MyBuffers[ ulLoop ].ulUserParm = ulLoop;

   }
```

MCI_BUFFER - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_CAPTURE

MCI_CAPTURE Parameter - ulParam1

ulParam1 (ULONG)
    This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CAPTURE_RECT
: Indicates that a region of the screen to be captured is provided in the *rect* field of the MCI_CAPTURE_PARMS structure pointed to by *pParam2* .

MCI_CONVERT
: Specifies that the captured image data will be converted to the OS/2 bit-map format when it is saved to disk.

MCI_CAPTURE Parameter - pParam2

pParam2 (PMCI_CAPTURE_PARMS)
A pointer to an MCI_CAPTURE_PARMS data structure.

MCI_CAPTURE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM2 driver for this command.

MCIERR_INSTANCE_INACTIVE
: The device has been opened as shareable and is currently in use by another application.

MCIERR_OVLY_INVALID_RECT
: An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
: The requested action is not available. (For example, because video has been set off.)

MCI_CAPTURE - Description

This message requests the digital video device to capture the current movie frame and store it as an image device element.

Note: MCI_CAPTURE captures bit maps from movies rather than hardware. See MCI_GETIMAGEBUFFER for a description of how to capture from hardware with a video capture card.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CAPTURE_RECT
Indicates that a region of the screen to be captured is provided in the *rect* field of the MCI_CAPTURE_PARMS structure pointed to by *pParam2* .

MCI_CONVERT
Specifies that the captured image data will be converted to the OS/2 bit-map format when it is saved to disk.

pParam2 (PMCI_CAPTURE_PARMS)
A pointer to an MCI_CAPTURE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM2 driver for this command.

MCIERR_INSTANCE_INACTIVE
The device has been opened as shareable and is currently in use by another application.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available. (For example, because video has been set off.)

MCI_CAPTURE - Remarks

This command is not supported by all devices. Use the MCI_GETDEVCAPS command to determine whether the device supports MCI_CAPTURE.

Repeated capture commands overwrite the image in the device element buffer. If the application wants to transfer the image data to a permanent file, it can use the MCI_SAVE message with the MCI_DGV_SAVE_IMAGE_FILE flag set. If the application wants the image copied to its address space, it issues MCI_GETIMAGEBUFFER.

The captured image is retained as the device element. With overlay video devices implemented on dual-plane video hardware, the image is captured from the *video* or *image* layer.

The media control device can perform the following operations:

- Freeze the motion temporarily, if needed, to capture the image.
- Obtain image data from the device and place the data into the capture and restore buffer.
- Perform an "unfreeze" (if necessary) to return to the original state.

  It will *not* convert, translate, or change the data from the internal format into another format.

  If no rectangle is specified, the entire video image in the video window is captured.

MCI_CAPTURE - Related Messages

- MCI_GETIMAGEBUFFER

MCI_CAPTURE - Example Code

The following code illustrates how to cause a video device to capture the current video image and store it as an image device element.

```
MCI_CAPTURE_PARMS mciCaptureParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;

/* Without a rectangle */
memset (&mciCaptureParms, 0x00, sizeof (MCI_CAPTURE_PARMS));
mciCaptureParms.hwndCallback = hwndNotify;
mciCaptureParms.rect       = 0;

ulReturn = mciSendCommand(usDeviceID, MCI_CAPTURE,
               MCI_WAIT,
               (PVOID)&mciCaptureParms,
               usUserParm);

/* With a rectangle */
memset (&mciCaptureParms, 0x00, sizeof (MCI_CAPTURE_PARMS));
mciCaptureParms.hwndCallback = hwndNotify;
mciCaptureParms.rect.xLeft   = ulX1;
mciCaptureParms.rect.yBottom = ulY1;
mciCaptureParms.rect.xRight  = ulX2;
mciCaptureParms.rect.yTop    = ulY2;


ulReturn = mciSendCommand(usDeviceID, MCI_CAPTURE,
               MCI_WAIT | MCI_CAPTURE_RECT,
               (PVOID)&mciCaptureParms,
               usUserParm);
```

MCI_CAPTURE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_CLOSE

MCI_CLOSE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CLOSE_EXIT
This flag is recognized and accepted by media control drivers (MCDs); however, it should only be sent by the Media Device Manager (MDM). This flag informs the MCD that this particular close operation is coming from an exit list routine. When an MCD receives this, it will terminate in the usual way. All other threads have been terminated. When this flag is received, the MCD must assume that the current thread is the only thread in its process.

MCI_CLOSE Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to a default media control interface parameter data structure.

MCI_CLOSE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCI_CLOSE - Description

This message requests that the current media device instance be closed and all resources associated with it be released.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CLOSE_EXIT
This flag is recognized and accepted by media control drivers (MCDs); however, it should only be sent by the Media Device Manager (MDM). This flag informs the MCD that this particular close operation is coming from an exit list routine. When an MCD receives this, it will terminate in the usual way. All other threads have been terminated. When this flag is received, the MCD must assume that the current thread is the only thread in its process.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to a default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID

Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCI_CLOSE - Example Code

The following code illustrates how to close a device context.

```
USHORT usDeviceID;                        /* Device ID              */
MCI_GENERIC_PARMS mciGenericParms;        /* Generic message
                                             parms structure        */

                                          /* Close a device context */
mciSendCommand( usDeviceID,               /* Device ID to close     */
                MCI_CLOSE,                /* MCI close message      */
                MCI_WAIT,                 /* Flag for this message  */
                (PVOID) &mciGenericParms,/* Data structure         */
                0);                       /* No user parameter      */
```

MCI_CLOSE - Topics

Select an item:
Description
Returns
Example Code
Glossary

MCI_CONNECTION

MCI_CONNECTION Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is

completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERY_CONNECTION
: Indicates that the media driver must return the device ID of the connected device in the *usToDeviceID* field. The MCI_CONNECTOR_TYPE and MCI_CONNECTOR_INDEX flags specify parameters that identify the desired connector. Once the device ID is obtained, an application can send messages directly to the connected device to obtain advanced functionality not directly provided by the original device. If no connection exists, MCIERR_NO_CONNECTION is returned.

MCI_CONNECTOR_TYPE
: Indicates that the *ulConnectorType* field specifies a connector type for the primary device. When this flag is used, the *ulConnectorIndex* field is interpreted as a relative index rather than an absolute index. The following connector types are currently defined:

  MCI_MIDI_STREAM_CONNECTOR
  : Digital input or output for the sequencer device. This data typically is streamed to an amplifier mixer device.

  MCI_CD_STREAM_CONNECTOR
  : Digital output for a CD audio device capable of reading the data directly off a disc. The data typically is streamed to an amplifier mixer device.

  MCI_WAVE_STREAM_CONNECTOR
  : Digital input or output for the waveform audio device. The data typically is streamed to an amplifier mixer device.

  MCI_XA_STREAM_CONNECTOR
  : Digital output for the CD XA device. The data typically is streamed to an amplifier mixer device.

  MCI_AMP_STREAM_CONNECTOR
  : Digital input or output for an amplifier mixer device.

  MCI_HEADPHONES_CONNECTOR
  : The connector on the device that is typically used to attach headphones to the device.

  MCI_SPEAKERS_CONNECTOR
  : The connector on the device that is typically used to attach speakers to the device.

  MCI_MICROPHONE_CONNECTOR
  : The connector on the device that is typically used to attach a microphone to the device.

  MCI_LINE_IN_CONNECTOR
  : The connector on the device that is typically used to provide line level input to the device.

  MCI_LINE_OUT_CONNECTOR
  : The connector on the device that is typically used to provide line level output from the device.

  MCI_VIDEO_IN_CONNECTOR
  : The connector on the device that is typically used to provide video input to the device.

  MCI_VIDEO_OUT_CONNECTOR
  : The connector on the device that is typically used to provide video output from the device.

  MCI_UNIVERSAL_CONNECTOR
  : A connector on a device that does not fall into any of the other categories. This connector can be used to access device-dependent function. The manufacturer of the device should define the exact use of this connector.

MCI_CONNECTOR_INDEX
: Indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified then an index of 1 is assumed.

MCI_CONNECTOR_ALIAS
: Indicates that the *pszAlias* field contains an alias for the device instance connected to the specified connector. If the alias already exists for another device, the error MCIERR_DUPLICATE_ALIAS is returned. If the connected to device already has an alias, the error MCIERR_CANNOT_ADD_ALIAS is returned. The primary purpose of this function is to permit access to connected devices through the string interface.

MCI_CONNECTION Parameter - pParam2

pParam2 (PMCI_CONNECTION_PARMS)
A pointer to the MCI_CONNECTION_PARMS data structure.

MCI_CONNECTION Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	The function is successful.
MCIERR_ALREADY_CONNECTED	A connection already exists for the specified connector.
MCIERR_INVALID_CONNECTION	Connection between the specified connection types is invalid.
MCIERR_CANNOT_ADD_ALIAS	The alias was not added.
MCIERR_DUPLICATE_ALIAS	The alias already exists.
MCIERR_NO_CONNECTION	No connection exists for the queried connector.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INVALID_DEVICE_ORDINAL	The device ordinal given is invalid.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_CONN_TYPE	This device does not support the given connector type.
MCIERR_INVALID_CONNECTOR_TYPE	The given connector type is invalid.
MCIERR_INVALID_CONNECTOR_INDEX	Invalid connector index given.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.

MCI_CONNECTION - Description

This message requests the device ID of a connected device instance. An alias also can be assigned to the connected device to facilitate the sending of string commands to that device.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERY_CONNECTION
Indicates that the media driver must return the device ID of the connected device in the *usToDeviceID* field. The MCI_CONNECTOR_TYPE and MCI_CONNECTOR_INDEX flags specify parameters that identify the desired connector. Once the device ID is obtained, an application can send messages directly to the connected device to obtain advanced functionality not directly provided by the original device. If no connection exists, MCIERR_NO_CONNECTION is returned.

MCI_CONNECTOR_TYPE
Indicates that the *ulConnectorType* field specifies a connector type for the primary device. When this flag is used, the *ulConnectorIndex* field is interpreted as a relative index rather than an absolute index. The following connector types are currently defined:

MCI_MIDI_STREAM_CONNECTOR
Digital input or output for the sequencer device. This data typically is streamed to an amplifier mixer device.

MCI_CD_STREAM_CONNECTOR
Digital output for a CD audio device capable of reading the data directly off a disc. The data typically is streamed to an amplifier mixer device.

MCI_WAVE_STREAM_CONNECTOR
Digital input or output for the waveform audio device. The data typically is streamed to an amplifier mixer device.

MCI_XA_STREAM_CONNECTOR
Digital output for the CD XA device. The data typically is streamed to an amplifier mixer device.

MCI_AMP_STREAM_CONNECTOR
Digital input or output for an amplifier mixer device.

MCI_HEADPHONES_CONNECTOR
The connector on the device that is typically used to attach headphones to the device.

MCI_SPEAKERS_CONNECTOR
The connector on the device that is typically used to attach speakers to the device.

MCI_MICROPHONE_CONNECTOR
The connector on the device that is typically used to attach a microphone to the device.

MCI_LINE_IN_CONNECTOR
The connector on the device that is typically used to provide line level input to the device.

MCI_LINE_OUT_CONNECTOR
The connector on the device that is typically used to provide line level output from the device.

MCI_VIDEO_IN_CONNECTOR
The connector on the device that is typically used to provide video input to the device.

MCI_VIDEO_OUT_CONNECTOR
The connector on the device that is typically used to provide video output from the device.

MCI_UNIVERSAL_CONNECTOR
A connector on a device that does not fall into any of the other categories. This connector can be used to access device-dependent function. The manufacturer of the device should define the exact use of this connector.

MCI_CONNECTOR_INDEX
: Indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified then an index of 1 is assumed.

MCI_CONNECTOR_ALIAS
: Indicates that the *pszAlias* field contains an alias for the device instance connected to the specified connector. If the alias already exists for another device, the error MCIERR_DUPLICATE_ALIAS is returned. If the connected to device already has an alias, the error MCIERR_CANNOT_ADD_ALIAS is returned. The primary purpose of this function is to permit access to connected devices through the string interface.

pParam2 (PMCI_CONNECTION_PARMS)
A pointer to the MCI_CONNECTION_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	The function is successful.
MCIERR_ALREADY_CONNECTED	A connection already exists for the specified connector.
MCIERR_INVALID_CONNECTION	Connection between the specified connection types is invalid.
MCIERR_CANNOT_ADD_ALIAS	The alias was not added.
MCIERR_DUPLICATE_ALIAS	The alias already exists.
MCIERR_NO_CONNECTION	No connection exists for the queried connector.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INVALID_DEVICE_ORDINAL	The device ordinal given is invalid.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_CONN_TYPE	This device does not support the given connector type.
MCIERR_INVALID_CONNECTOR_TYPE	The given connector type is invalid.
MCIERR_INVALID_CONNECTOR_INDEX	Invalid connector index given.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.

MCI_CONNECTION - Remarks

It is recommended that all applications refer to connectors using the MCI_CONNECTOR_TYPE flag. This provides device independence from differences in connector numbering for various hardware devices. Additionally, the MCI_CONNECTOR_INDEX flag can be used to address different connectors of the same type.

If only the MCI_CONNECTOR_INDEX flag is used, the referenced connector is device dependent. The connector type of a particular connector index, as well as the number of connectors, can be retrieved using the MCI_CONNECTORINFO or MCI_SYSINFO messages.

For a list of connector types which are supported by various device types, see the **Remarks** section for MCI_CONNECTORINFO.

MCI_CONNECTION - Default Processing

If MCI_CONNECTOR_INDEX is not specified, the connector number defaults to 1. If MCI_CONNECTOR_TYPE is not specified, then an absolute connector number is assumed.

MCI_CONNECTION - Related Messages

- MCI_CONNECTOR
- MCI_CONNECTORINFO

MCI_CONNECTION - Example Code

The following code illustrates how to query the device ID of the ampmix device, which is consuming the digital audio data stream from a waveaudio device.

```
USHORT                     usWaveDeviceID;
USHORT                     usAmpDeviceID;
MCI_CONNECTION_PARMS       connectionparms;

connectionparms.ulConnectorType = MCI_WAVE_STREAM_CONNECTOR;

                                   /* Get the Amp/Mixer device ID */

mciSendCommand ( usWaveDeviceID,           /* WaveAudio device ID */
   MCI_CONNECTION,                         /* CONNECTION message  */
   MCI_QUERY_CONNECTION | MCI_WAIT,        /* Flags for this msg  */
          (PVOID) &connectionparms,        /* Data structure      */
           0 );                            /* No user parameter   */

usAmpDeviceID = connectionparms.usToDeviceID;
                                           /* Device ID amp mixer */
```

MCI_CONNECTION - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Related Messages
Example Code
Glossary

MCI_CONNECTOR

MCI_CONNECTOR Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ENABLE_CONNECTOR
Enables input or output through the specified connector.

MCI_DISABLE_CONNECTOR
Disables input or output through the specified connector.

MCI_QUERY_CONNECTOR_STATUS
Queries the status of the specified connector and returns the result in the *ulReturn* field of the parameter data structure pointed to by *pParam2*. The possible states are enabled or disabled.

MCI_CONNECTOR_TYPE
Indicates that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulConnectorIndex* field is interpreted as a relative index rather than an absolute index. The following connector types are currently defined:

MCI_MIDI_STREAM_CONNECTOR
Digital input or output for the sequencer device. This data typically is streamed to an amplifier mixer device.

MCI_CD_STREAM_CONNECTOR
Digital output for a CD audio device capable of reading the data directly off a disc. The data typically is streamed to an amplifier mixer device.

MCI_WAVE_STREAM_CONNECTOR
Digital input or output for the waveform audio device. The data typically is streamed to an amplifier mixer device.

This connector type is not supported by the digital video MCD.

MCI_XA_STREAM_CONNECTOR
Digital output for the CD XA device. The data typically is streamed to an amplifier mixer device.

MCI_AMP_STREAM_CONNECTOR
Digital input or output for an amplifier mixer device.

MCI_HEADPHONES_CONNECTOR
The connector on the device that is typically used to attach headphones to the device.

MCI_SPEAKERS_CONNECTOR
The connector on the device that is typically used to attach speakers to the device.

MCI_MICROPHONE_CONNECTOR
The connector on the device that is typically used to attach a microphone to the device.

MCI_LINE_IN_CONNECTOR
The connector on the device that is typically used to provide line level input to the device.

MCI_LINE_OUT_CONNECTOR
The connector on the device that is typically used to provide line level output from the device.

MCI_AUDIO_IN_CONNECTOR
The connector on the device that is typically used to provide audio input to the device.

MCI_AUDIO_OUT_CONNECTOR
The connector on the device that is typically used to provide audio output from the device.

MCI_VIDEO_IN_CONNECTOR
The connector on the device that is typically used to provide video input to the device.

MCI_VIDEO_OUT_CONNECTOR
The connector on the device that is typically used to provide video output from the device.

MCI_UNIVERSAL_CONNECTOR
A connector on a device that does not fall into any of the other categories. This connector type can be used to access a device-dependent function. The manufacturer of the device should define the exact use of this connector.

MCI_CONNECTOR_INDEX
Indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified then an index of 1 is assumed.

MCI_CONNECTOR Parameter - pParam2

pParam2 (PMCI_CONNECTOR_PARMS)
A pointer to the MCI_CONNECTOR_PARMS data structure.

MCI_CONNECTOR Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags not compatible.

MCIERR_INSTANCE_INACTIVE
: Instance inactive.

MCIERR_INVALID_CONNECTOR_INDEX
: Invalid connector index.

MCIERR_INVALID_CONNECTOR_TYPE
: Invalid connector type given.

MCIERR_UNSUPPORTED_CONN_TYPE
: Connector type is not supported by this device.

MCIERR_CANNOT_MODIFY_CONNECTOR
: Cannot enable or disable this connector.

MCI_CONNECTOR - Description

This message is used to enable, disable or query the status of a particular connector for a device instance. The connector can be specified either absolutely or as a relative offset within a specified connector type.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ENABLE_CONNECTOR
: Enables input or output through the specified connector.

MCI_DISABLE_CONNECTOR
: Disables input or output through the specified connector.

MCI_QUERY_CONNECTOR_STATUS
: Queries the status of the specified connector and returns the result in the *ulReturn* field of the parameter data structure pointed to by *pParam2*. The possible states are enabled or disabled.

MCI_CONNECTOR_TYPE
: Indicates that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulConnectorIndex* field is interpreted as a relative index rather than an absolute index. The following connector types are currently defined:

MCI_MIDI_STREAM_CONNECTOR
: Digital input or output for the sequencer device. This data typically is streamed to an amplifier mixer device.

MCI_CD_STREAM_CONNECTOR
: Digital output for a CD audio device capable of reading the data directly off a disc. The data typically is streamed to an amplifier mixer device.

MCI_WAVE_STREAM_CONNECTOR
: Digital input or output for the waveform audio device. The data typically is streamed to an amplifier mixer device.

This connector type is not supported by the digital video MCD.

MCI_XA_STREAM_CONNECTOR
Digital output for the CD XA device. The data typically is streamed to an amplifier mixer device.

MCI_AMP_STREAM_CONNECTOR
Digital input or output for an amplifier mixer device.

MCI_HEADPHONES_CONNECTOR
The connector on the device that is typically used to attach headphones to the device.

MCI_SPEAKERS_CONNECTOR
The connector on the device that is typically used to attach speakers to the device.

MCI_MICROPHONE_CONNECTOR
The connector on the device that is typically used to attach a microphone to the device.

MCI_LINE_IN_CONNECTOR
The connector on the device that is typically used to provide line level input to the device.

MCI_LINE_OUT_CONNECTOR
The connector on the device that is typically used to provide line level output from the device.

MCI_AUDIO_IN_CONNECTOR
The connector on the device that is typically used to provide audio input to the device.

MCI_AUDIO_OUT_CONNECTOR
The connector on the device that is typically used to provide audio output from the device.

MCI_VIDEO_IN_CONNECTOR
The connector on the device that is typically used to provide video input to the device.

MCI_VIDEO_OUT_CONNECTOR
The connector on the device that is typically used to provide video output from the device.

MCI_UNIVERSAL_CONNECTOR
A connector on a device that does not fall into any of the other categories. This connector type can be used to access a device-dependent function. The manufacturer of the device should define the exact use of this connector.

MCI_CONNECTOR_INDEX
Indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified then an index of 1 is assumed.

pParam2 (PMCI_CONNECTOR_PARMS)
A pointer to the MCI_CONNECTOR_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_INVALID_CONNECTOR_INDEX
Invalid connector index.

MCIERR_INVALID_CONNECTOR_TYPE
Invalid connector type given.

MCIERR_UNSUPPORTED_CONN_TYPE
Connector type is not supported by this device.

MCIERR_CANNOT_MODIFY_CONNECTOR
Cannot enable or disable this connector.

MCI_CONNECTOR - Remarks

It is recommended that all applications refer to connectors using the MCI_CONNECTOR_TYPE flag. This provides device independence from differences in connector numbering for various devices. Additionally, the MCI_CONNECTOR_INDEX flag can be used to address more than one connector of the same type.

If only the MCI_CONNECTOR_INDEX flag is used, the referenced connector is device dependent. The connector type of a particular connector index, as well as the number of connectors, can be retrieved using the MCI_CONNECTORINFO message.

The amplifier-mixer device for the M-Audio Adapter does not have a *headphone* connector.

Disabling a connector on a device can terminate an active command.

For a list of connector types which are supported by various device types, see the **Remarks** section for MCI_CONNECTORINFO.

MCI_CONNECTOR - Default Processing

If MCI_CONNECTOR_INDEX is not specified, the connector index defaults to 1. If MCI_CONNECTOR_TYPE is not specified, an absolute index is assumed.

MCI_CONNECTOR - Example Code

The following code illustrates how to enable microphone input on an audio device.

```
USHORT                   usAmpDeviceID;
MCI_CONNECTOR_PARMS      connectorparms;

connectorparms.ulConnectorType = MCI_MICROPHONE_CONNECTOR;

                                      /* Enable microphone input on */
                                      /* the audio device           */

mciSendCommand (usAmpDeviceID,        /* Amp/mixer device ID        */
 MCI_CONNECTOR,                       /* CONNECTOR message          */
 MCI_ENABLE_CONNECTOR | MCI_CONNECTOR_TYPE | MCI _WAIT,
                                      /* Flags for this message     */
 (PVOID) &connectorparms,             /* Data structure             */
 0 );                                 /* No user parm               */
```

MCI_CONNECTOR - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Example Code
Glossary

MCI_CONNECTORINFO

MCI_CONNECTORINFO Parameter - ulParam1

ulParam1 (ULONG)
The parameter can contain any the following flags with the following exceptions: The MCI_ENUMERATE_CONNECTORS, MCI_QUERY_CONNECTOR_TYPE, and MCI_QUERY_VALID_CONNECTION flags are mutually exclusive. In addition, MCI_ENUMERATE_CONNECTORS and MCI_CONNECTOR_INDEX are mutually exclusive, and the error MCIERR_FLAGS_NOT_COMPATIBLE is returned if these flags are used together.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CONNECTOR_TYPE
This flag indicates that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used then the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_CONNECTOR_INDEX
This flag indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified, an index of 1 is assumed.

MCI_QUERY_CONNECTOR_TYPE
This flag returns connector type in the *ulReturn* field. To specify the connector to query, use the MCI_CONNECTOR_INDEX flag.

MCI_ENUMERATE_CONNECTORS
This flag returns the number of connectors for the given device. If the MCI_CONNECTOR_TYPE flag is also specified, the number of connectors for the specified type is returned. The value is returned in the *ulReturn* field.

MCI_QUERY_VALID_CONNECTION
This flag determines if the specified connection is possible. MCI_TRUE is returned if the connector types specified in the *ulConnectorType* and *ulToConnectorType* fields are compatible, resulting in a valid connection. Otherwise, MCI_FALSE is returned.

MCI_TO_CONNECTOR_TYPE
This flag specifies that the connector type (*ulToConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_CONNECTORINFO Parameter - pParam2

pParam2 (PMCI_CONNECTORINFO_PARMS)
A pointer to the MCI_CONNECTORINFO_PARMS data structure.

MCI_CONNECTORINFO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_INVALID_DEVICE_ORDINAL	The device ordinal given is invalid.
MCIERR_INVALID_DEVICE_TYPE	The device type given is invalid.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_INVALID_FLAG	Given flag is invalid.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_INVALID_CONNECTOR_TYPE	The given connector type is invalid.
MCIERR_INVALID_CONNECTOR_INDEX	Invalid connector index given.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.

MCI_CONNECTORINFO - Description

This message is used to determine the total number of connectors on a device, the number of connectors of a specific type, the type of each connector, and whether or not a particular type of connection is valid for a connector.

ulParam1 (ULONG)

The parameter can contain any the following flags with the following exceptions: The MCI_ENUMERATE_CONNECTORS, MCI_QUERY_CONNECTOR_TYPE, and MCI_QUERY_VALID_CONNECTION flags are mutually exclusive. In addition, MCI_ENUMERATE_CONNECTORS and MCI_CONNECTOR_INDEX are mutually exclusive, and the error MCIERR_FLAGS_NOT_COMPATIBLE is returned if these flags are used together.

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CONNECTOR_TYPE
: This flag indicates that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used then the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_CONNECTOR_INDEX
: This flag indicates that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified, an index of 1 is assumed.

MCI_QUERY_CONNECTOR_TYPE
: This flag returns connector type in the *ulReturn* field. To specify the connector to query, use the MCI_CONNECTOR_INDEX flag.

MCI_ENUMERATE_CONNECTORS
: This flag returns the number of connectors for the given device. If the MCI_CONNECTOR_TYPE flag is also specified, the number of connectors for the specified type is returned. The value is returned in the *ulReturn* field.

MCI_QUERY_VALID_CONNECTION
: This flag determines if the specified connection is possible. MCI_TRUE is returned if the connector types specified in the *ulConnectorType* and *ulToConnectorType* fields are compatible, resulting in a valid connection. Otherwise, MCI_FALSE is returned.

MCI_TO_CONNECTOR_TYPE
: This flag specifies that the connector type (*ulToConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

pParam2 (PMCI_CONNECTORINFO_PARMS)

A pointer to the MCI_CONNECTORINFO_PARMS data structure.

rc (ULONG)

Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ORDINAL
: The device ordinal given is invalid.

MCIERR_INVALID_DEVICE_TYPE
: The device type given is invalid.

MCIERR_MISSING_FLAG
: A required flag is missing.

MCIERR_INVALID_FLAG
: Given flag is invalid.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_INVALID_CONNECTOR_TYPE
: The given connector type is invalid.

MCIERR_INVALID_CONNECTOR_INDEX
: Invalid connector index given.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_CONNECTORINFO - Remarks

This message does not require a device instance to be open.

The following is a list of connector types supported by each OS/2 multimedia device:

Amplifier Mixer Device

MCI_AMP_STREAM_CONNECTOR
MCI_HEADPHONES_CONNECTOR
MCI_LINE_IN_CONNECTOR
MCI_LINE_OUT_CONNECTOR
MCI_MICROPHONE_CONNECTOR
MCI_SPEAKERS_CONNECTOR

CD Audio Device

MCI_CD_STREAM_CONNECTOR
MCI_HEADPHONES_CONNECTOR

CD/XA Device

MCI_XA_STREAM_CONNECTOR

Sequencer Device

MCI_MIDI_STREAM_CONNECTOR

The sequencer also understands the following connector types and will attempt to access the connector on its associated amplifier mixer device.

MCI_HEADPHONES_CONNECTOR
MCI_LINE_OUT_CONNECTOR
MCI_SPEAKERS_CONNECTOR

Waveform Audio Device

MCI_WAVE_STREAM_CONNECTOR

The waveform audio device also understands the following connector types and will attempt to access the connector on its associated amplifier mixer device.

MCI_HEADPHONES_CONNECTOR
MCI_LINE_IN_CONNECTOR
MCI_LINE_OUT_CONNECTOR
MCI_MICROPHONE_CONNECTOR
MCI_SPEAKERS_CONNECTOR

Videodisc Device

MCI_LINE_OUT_CONNECTOR
MCI_VIDEO_OUT_CONNECTOR

Digital Video Device

MCI_WAVE_STREAM_CONNECTOR

The digital video device also understands the following connector types and will attempt to access the connector on its associated amplifier mixer device:

MCI_HEADPHONES_CONNECTOR
MCI_LINE_IN_CONNECTOR
MCI_LINE_OUT_CONNECTOR
MCI_MICROPHONE_CONNECTOR
MCI_SPEAKERS_CONNECTOR
MCI_VIDEO_IN_CONNECTOR
MCI_VIDEO_OUT_CONNECTOR

MCI_VIDEO_IN_CONNECTOR and MCI_VIDEO_OUT_CONNECTOR connector types are only supported in recording environments.

MCI_CONNECTORINFO - Default Processing

If the MCI_CONNECTOR_INDEX flag is not specified, the connector index will default to 1.

MCI_CONNECTORINFO - Related Messages

- MCI_CONNECTOR

MCI_CONNECTORINFO - Example Code

The following code illustrates how to determine whether a device has microphone input capability.

```
            /* Determine if amp/mixer device has a microphone input */

MCI_CONNECTORINFO_PARMS conninfoparms;
ULONG rc;
ULONG NumMicConns;

conninfoparms.ulDeviceTypeID = MCI_DEVTYPE_AUDIO_AMPMIX;
conninfoparms.ulConnectorType = MCI_MICROPHONE_CONNECTOR;


rc = mciSendCommand (0,           /* Ignored field              */
  MCI_CONNECTORINFO,              /* Connectorinfo message      */
  MCI_ENUMERATE_CONNECTORS | MCI_WAIT | MCI_CONNECTOR_TYPE,
                                  /* Flags for this message     */
  (PVOID) &conninfoparms,         /* Data structure             */
  0);                             /* No user parm               */
if (LOUSHORT(rc) == MCIERR_SUCCESS)
   {
    NumMicConns = conninfoparms.ulReturn; /* Return information */
  }
```

MCI_CONNECTORINFO - Topics

Select an item:

MCI_COPY

MCI_COPY Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.
MCI_FROM	The beginning position of a copy from a file. The position of the media will either be the position specified in MCI_FROM or the previous position if MCI_FROM is not specified.
MCI_TO	The ending position of a copy from a file.
MCI_FROM_BUFFER	Places information from a buffer into the clipboard. If this flag is not specified, the file is used.
MCI_TO_BUFFER	Places information from a file into a buffer. If this flag is not specified, the clipboard is used.

MCI_COPY Parameter - pParam2

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

MCI_COPY Return Value - rc

rc (ULONG)

Return codes indicating success or type of failure:

MCIERR_SUCCESS
Copy was successful.

MCIERR_INVALID_BUFFER
Buffer was too small to hold data.

MCIERR_OUTOFRANGE
The units are out of the range.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to perform the operation.

MCIERR_CLIPBOARD_ERROR
A problem with the clipboard occurred.

MCI_COPY - Description

This message copies the specified range of data from the device file to the clipboard or application buffer. The position of the media remains the same as prior to the copy operation.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
The beginning position of a copy from a file. The position of the media will either be the position specified in MCI_FROM or the previous position if MCI_FROM is not specified.

MCI_TO
The ending position of a copy from a file.

MCI_FROM_BUFFER
Places information from a buffer into the clipboard. If this flag is not specified, the file is used.

MCI_TO_BUFFER

Places information from a file into a buffer. If this flag is not specified, the clipboard is used.

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
Copy was successful.

MCIERR_INVALID_BUFFER
Buffer was too small to hold data.

MCIERR_OUTOFRANGE
The units are out of the range.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to perform the operation.

MCIERR_CLIPBOARD_ERROR
A problem with the clipboard occurred.

MCI_COPY - Remarks

MCI_COPY copies the range of media data specified by the *ulFrom* and *ulTo* fields in the MCI_EDIT_PARMS data structure to an application-supplied buffer or the system clipboard. If the *pBuff* field of the data structure contains a pointer and the MCI_TO_BUFFER flag is specified, the data is copied to a buffer. If the MCI_FROM_BUFFER flag is specified, the information is copied from the buffer to the clipboard.

The units of the MCI_FROM and MCI_TO parameters are interpreted in the currently selected time format. If neither MCI_FROM nor MCI_TO are specified, the range is assumed from the current file position to the end of the file. The difference between MCI_FROM and MCI_TO must be greater than zero, otherwise an error is returned.

Edited Audio/Video Interleaved (AVI) movie files cannot always be saved with their original name after a copy operation. If the clipboard contains a reference to data that would be erased during saving or if another instance of the digital video device has a pending paste operation which depends on this data, the file cannot be saved unless a new file name has been provided. If a new file name is not provided, MMIOERR_NEED_NEW_FILENAME is returned by the AVI I/O procedure and a temporary file is created to save the edited movie.

Note: AVI is the only video file format supporting editing commands.

If data is already in the clipboard, then it is overwritten. If a copy interrupts an in-progress operation, such as play, the operation is aborted and an MM_MCINOTIFY message is sent to the application.

If an invalid buffer length is passed in, *ulBufLen* is updated with the correct length.

Waveaudio Specific

If MCI_FROM_BUFFER or MCI_TO_BUFFER are used, the *pHeader* field of MCI_EDIT_PARMS must contain a pointer to an MMAUDIOHEADER structure. The *ulBufLen* field of MCI_EDIT_PARMS must be filled in.

MCI_COPY - Related Messages

- MCI_CUT
- MCI_DELETE
- MCI_PASTE
- MCI_REDO
- MCI_UNDO

MCI_COPY - Example Code

The following code illustrates how to copy the first five seconds of a file and place it in the clipboard.

```
USHORT          usDeviceID;
MCI_EDIT_PARMS  mep;

mep.hwndCallback = hwndMyWindow;
mep.ulFrom = 0;
mep.ulTo = 5000;

mciSendCommand( usDeviceID,
                MCI_COPY
                MCI_NOTIFY | MCI_FROM | MCI_TO,
                &mep,
                0 );
```

MCI_COPY - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_CUE

MCI_CUE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags with the following limitation. The MCI_CUE_INPUT and MCI_CUE_OUTPUT flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CUE_INPUT
This flag cues or prerolls the device instance for input or recording.

MCI_CUE_OUTPUT
This flag cues or prerolls the device instance for output or playback.

Digital Video Extensions

The following additional flags apply to digital video devices. These flags are only valid when cueing the device instance for output. The MCI_NOSHOW and MCI_SHOW flags are mutually exclusive.

MCI_NOSHOW
This flag causes the window to be hidden while the cue operation is performed. This is the default. If MCI_TO is not also specified, the media position will remain unchanged.

MCI_SHOW
This flag causes the window to be displayed while the cue operation is performed. If MCI_TO is not also specified, the current frame is displayed and the media position will advance by one (frame).

MCI_TO
This flag enables seeking to a specific location in the media while cueing the device for playback. If this flag is specified, the *ulTo* field of MCI_SEEK_PARMS indicates the ending position of the seek operation. If the *ulTo* position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

Wave Audio Extensions

The following additional flags apply to wave audio devices. The MCI_WAVE_INPUT and MCI_WAVE_OUTPUT flags are mutually exclusive.

MCI_WAVE_INPUT
This flag cues or prerolls the device instance for input or recording.

MCI_WAVE_OUTPUT
This flag cues or prerolls the device instance for output or playback.

MCI_CUE Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_SEEK_PARMS
A pointer to the MCI_SEEK_PARMS structure.

MCI_CUE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_CUE - Description

This message prompts a device instance to ready itself (preroll) for a subsequent playback or recording message with minimum delay.

ulParam1 (ULONG)
This parameter can contain any of the following flags with the following limitation. The MCI_CUE_INPUT and MCI_CUE_OUTPUT flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CUE_INPUT
This flag cues or prerolls the device instance for input or recording.

MCI_CUE_OUTPUT
This flag cues or prerolls the device instance for output or playback.

Digital Video Extensions

The following additional flags apply to digital video devices. These flags are only valid when cueing the device instance for output. The MCI_NOSHOW and MCI_SHOW flags are mutually exclusive.

MCI_NOSHOW
This flag causes the window to be hidden while the cue operation is performed. This is the default. If MCI_TO is not also specified, the media position will remain unchanged.

MCI_SHOW
This flag causes the window to be displayed while the cue operation is performed. If MCI_TO is not also specified, the current frame is displayed and the media position will advance by one (frame).

MCI_TO
This flag enables seeking to a specific location in the media while cueing the device for playback. If this flag is specified, the *ulTo* field of MCI_SEEK_PARMS indicates the ending position of the seek operation. If the *ulTo* position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

Wave Audio Extensions

The following additional flags apply to wave audio devices. The MCI_WAVE_INPUT and MCI_WAVE_OUTPUT flags are mutually exclusive.

MCI_WAVE_INPUT
This flag cues or prerolls the device instance for input or recording.

MCI_WAVE_OUTPUT
This flag cues or prerolls the device instance for output or playback.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_SEEK_PARMS
A pointer to the MCI_SEEK_PARMS structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_CUE - Remarks

The preroll characteristics of the device can be obtained with MCI_GETDEVCAPS. On devices that require a file, the file must be loaded before the MCI_CUE command is issued; otherwise, MCIERR_FILE_NOT_FOUND is returned. If no flags are specified then the device instance is queued for output by default. MCI_CUE_INPUT is only supported on devices that support recording.

MCI_CUE - Related Messages

- MCI_PLAY
- MCI_RECORD

MCI_CUE - Example Code

The following code illustrates how to cue a device instance for playback and wait for completion.

```
/* Cue the device for playback (output), and wait for completion */

USHORT    usDeviceID;
HWND      hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms;        /* Generic message
                                             parms structure        */


                 /* Assign hwndCallback the handle to the PM window */

mciGenericParms.hwndCallback =  hwndMyWindow;

mciSendCommand( usDeviceID,               /* Device ID              */
   MCI_CUE,                               /* MCI cue message        */
   MCI_WAIT | MCI_CUE_OUTPUT,             /* Standard flags         */
   (PVOID)&mciGenericParms,               /* Generic structure      */
   0 );                                   /* No user parm           */
```

MCI_CUE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_CUT

MCI_CUT Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.
MCI_FROM	The beginning position of a cut operation. The position of the media is either the position specified in the *ulFrom* field or the previous position if MCI_FROM is not specified.
MCI_TO	The ending position of a cut operation.
MCI_TO_BUFFER	Place the data from a file into an application-supplied buffer. If this flag is not specified, then the clipboard is used.

MCI_CUT Parameter - pParam2

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

MCI_CUT Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	Cut was successful.
MCIERR_INVALID_BUFFER	Buffer too small to hold data.
MCIERR_CANNOT_WRITE	The file was not opened with write access.
MCIERR_OUTOFRANGE	The units are out of the range.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to perform the requested operation.

MCIERR_CLIPBOARD_ERROR
An error occurred while attempting to access the clipboard.

MCI_CUT - Description

This message removes the specified range of data from the device element and places it in the system clipboard or application-supplied buffer. The position of the media after a cut command is the FROM position, if MCI_FROM is specified. If MCI_FROM is not specified, the current position is used.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
The beginning position of a cut operation. The position of the media is either the position specified in the *ulFrom* field or the previous position if MCI_FROM is not specified.

MCI_TO
The ending position of a cut operation.

MCI_TO_BUFFER
Place the data from a file into an application-supplied buffer. If this flag is not specified, then the clipboard is used.

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
Cut was successful.

MCIERR_INVALID_BUFFER
Buffer too small to hold data.

MCIERR_CANNOT_WRITE
The file was not opened with write access.

MCIERR_OUTOFRANGE
The units are out of the range.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to perform the requested operation.

MCIERR_CLIPBOARD_ERROR
An error occurred while attempting to access the clipboard.

MCI_CUT - Remarks

If MCI_TO_BUFFER is specified and the buffer is not large enough to hold the data, then the error MCIERR_INVALID_BUFFER is returned.

The units of the MCI_FROM and MCI_TO parameters are interpreted in the currently selected time format. If neither MCI_FROM nor MCI_TO are specified, the range is assumed from the current position to the end of the file.

The difference between MCI_FROM and MCI_TO must be greater than zero; otherwise, an error is returned.

If data is already in the clipboard, then it is overwritten. If a cut interrupts an in-progress operation, such as play, the operation is aborted and an MM_MCINOTIFY message is sent to the application.

Edited Audio/Video Interleaved (AVI) movie files cannot always be saved with their original name after the cut operation. If the clipboard contains a reference to data that would be erased during saving or if another instance of the digital video device has a pending paste operation which depends on this data, the file cannot be saved unless a new file name has been provided. If a new file name is not provided, MMIOERR_NEED_NEW_FILENAME is returned by the AVI I/O procedure and a temporary file is created to save the edited movie.

Note: AVI is the only video file format supporting editing commands.

Waveaudio Specific

If either MCI_FROM or MCI_TO begin in the middle of a digital audio sample, the wave audio device begins at the beginning of that sample. If MCI_FROM_BUFFER or MCI_TO_BUFFER are used, the *pHeader* field of MCI_EDIT_PARMS must contain a pointer to an MMAUDIOHEADER structure. The *ulBufLen* field of MCI_EDIT_PARMS must be filled in.

MCI_CUT - Related Messages

- MCI_COPY
- MCI_PASTE
- MCI_DELETE
- MCI_UNDO

- MCI_REDO

MCI_CUT - Example Code

The following code illustrates removing five seconds of a file.

```
    USHORT                usDeviceID;
    MCI_EDIT_PARMS        mep;

    mep.hwndCallback =  hwndMyWindow;
    mep.ulFrom = 0;
    mep.ulTo = 5000;

    mciSendCommand( usDeviceID,
                    MCI_CUT,
                    MCI_NOTIFY | MCI_FROM | MCI_TO,
                    &mep,
                    0 );
```

MCI_CUT - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_DEFAULT_CONNECTION

MCI_DEFAULT_CONNECTION Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERY_CONNECTION
: This flag specifies that the default connection associated with the indicated connector is to be returned in the *pszToDevice* , *ulToConnectorType* , and *ulToConnectorIndex* fields of the MCI_DEFAULT_CONNECTION_PARMS data structure.

MCI_MAKE_CONNECTION
: This flag specifies that a default connection is to be established between the current device and the *ulDeviceTypeID* field of the data structure pointed to by *pParam2* . The precise connectors on each device can be indicated using the associated connector type and index flags.

MCI_BREAK_CONNECTION
: This flag specifies that the default connection associated with the indicated connector is to be broken.

MCI_CONNECTOR_TYPE
: This flag specifies that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_CONNECTOR_INDEX
: This flag specifies that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified an index of 1 is assumed.

MCI_TO_CONNECTOR_TYPE
: This flag specifies that the connector type (*ulToConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulToConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_TO_CONNECTOR_INDEX
: This flag specifies that the *ulToConnectorIndex* field contains the connector index for the primary device. If this flag is not specified an index of 1 is assumed.

MCI_DEFAULT_CONNECTION Parameter - pParam2

pParam2 (PMCI_DEFAULT_CONNECTION_PARMS)
: A pointer to the MCI_DEFAULT_CONNECTION_PARMS data structure.

MCI_DEFAULT_CONNECTION Return Value - rc

rc (ULONG)
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_INSTANCE_INACTIVE
: The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make the device instance active.

MCIERR_MISSING_FLAG
: A required flag is missing.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_UNSUPPORTED_CONN_TYPE
: This device does not support the given connector type.

MCIERR_INVALID_CONNECTOR_TYPE
: The given connector type is invalid.

MCIERR_INVALID_CONNECTOR_INDEX
: Invalid connector index given.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCIERR_INVALID_FLAG
: Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCIERR_INVALID_CONNECTION
: An attempt was made to make an invalid connection.

MCIERR_NO_CONNECTION
: An attempt was made to break a nonexistent connection.

MCIERR_INVALID_DEVICE_ID
: A device ID is not valid.

MCIERR_INVALID_DEVICE_ORDINAL
: Invalid device ordinal given.

MCI_DEFAULT_CONNECTION - Description

This message is used to make, break, and query default connections between devices.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERY_CONNECTION
: This flag specifies that the default connection associated with the indicated connector is to be returned in the *pszToDevice* , *ulToConnectorType* , and *ulToConnectorIndex* fields of the MCI_DEFAULT_CONNECTION_PARMS data structure.

MCI_MAKE_CONNECTION
: This flag specifies that a default connection is to be established between the current device and the *ulDeviceTypeID* field of the data structure pointed to by *pParam2* . The precise connectors on each device can be indicated using the associated connector type and index flags.

MCI_BREAK_CONNECTION
: This flag specifies that the default connection associated with the indicated connector is to be broken.

MCI_CONNECTOR_TYPE
: This flag specifies that the connector type (*ulConnectorType* field) for the primary device is to be used for the query. When this flag is used the *ulConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_CONNECTOR_INDEX
: This flag specifies that the *ulConnectorIndex* field contains the connector index for the primary device. If this flag is not specified an index of 1 is assumed.

MCI_TO_CONNECTOR_TYPE
: This flag specifies that the connector type (*ulToConnectorType* field) for the primary device is to be used for the query. When this flag is used, the *ulToConnectorIndex* field is used as a relative index rather than an absolute index.

MCI_TO_CONNECTOR_INDEX
: This flag specifies that the *ulToConnectorIndex* field contains the connector index for the primary device. If this flag is not specified an index of 1 is assumed.

pParam2 (PMCI_DEFAULT_CONNECTION_PARMS)
A pointer to the MCI_DEFAULT_CONNECTION_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make the device instance active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_CONN_TYPE
This device does not support the given connector type.

MCIERR_INVALID_CONNECTOR_TYPE
The given connector type is invalid.

MCIERR_INVALID_CONNECTOR_INDEX
Invalid connector index given.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_CONNECTION
An attempt was made to make an invalid connection.

MCIERR_NO_CONNECTION
An attempt was made to break a nonexistent connection.

MCIERR_INVALID_DEVICE_ID
A device ID is not valid.

MCIERR_INVALID_DEVICE_ORDINAL
Invalid device ordinal given.

MCI_DEFAULT_CONNECTION - Remarks

Connector indexes start at index value 1. This message does not require a device to be opened.

For a list of connector types which are supported by various device types, see the MCI_CONNECTORINFO message.

MCI_DEFAULT_CONNECTION - Default Processing

If MCI_CONNECTOR_INDEX or MCI_TO_CONNECTOR flags are not specified, the associated connector index defaults to 1.

If MCI_CONNECTOR_TYPE or MCI_TO_CONNECTOR_TYPE flags are not specified, then the associated indexes are absolute.

MCI_DEFAULT_CONNECTION - Related Messages

- MCI_CONNECTION
- MCI_CONNECTOR
- MCI_CONNECTORINFO

MCI_DEFAULT_CONNECTION - Example Code

The following code illustrates how to determine the default connection for waveaudio.

```
MCI_DEFAULT_CONNECTION_PARMS    defaultconnparms;

defaultconnparms.ulConnectorType = MCI_WAVE_STREAM_CONNECTOR;
defaultconnparms.pszDevice = MCI_DEVTYPE_WAVEFORM_AUDIO_NAME;

                  /* Determine the default connection for waveaudio */

mciSendCommand ( 0,                   /* Ignore field              */
   MCI_DEFAULT_CONNECTION,            /* Default connection message */
   MCI_QUERY_CONNECTION | MCI_CONNECTOR_TYPE | MCI_WAIT,
                                      /* Flags for this message     */
   (PVOID) &defaultconnparms,         /* Data structure             */
   0 );                               /* No user parm               */

 /* Note: defaultconnparms.pszToDevice now contains the name of
    the device with default connection to the waveaudio (ampmixNN). */
```

MCI_DEFAULT_CONNECTION - Topics

Select an item:
Description
Returns
Remarks
Default Processing

Related Messages
Example Code
Glossary

MCI_DELETE

MCI_DELETE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.
MCI_FROM	The beginning position of a delete. The position of the media is either the position specified in the *ulFrom* field or the current position if MCI_FROM is not specified.
MCI_TO	The ending position of a delete operation. If MCI_TO is not specified, the end of the file is assumed to be the end of the range to be deleted.

MCI_DELETE Parameter - pParam2

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

MCI_DELETE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	Delete was successful.
MCIERR_CANNOT_WRITE	The file was not opened with write access.
MCIERR_OUTOFRANGE	

The units are out of the range.

MCIERR_INVALID_DEVICE_ID
: The device ID is not valid.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCIERR_INVALID_FLAG
: Flag is invalid (*ulParam1*).

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE
: The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
: Insufficient memory to perform the operation requested.

MCI_DELETE - Description

This message removes the specified range of data from the device file. The media position after a delete operation is the MCI_FROM position if used, or the previous position if MCI_FROM is not used.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
: The beginning position of a delete. The position of the media is either the position specified in the *ulFrom* field or the current position if MCI_FROM is not specified.

MCI_TO
: The ending position of a delete operation. If MCI_TO is not specified, the end of the file is assumed to be the end of the range to be deleted.

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: Delete was successful.

MCIERR_CANNOT_WRITE
: The file was not opened with write access.

MCIERR_OUTOFRANGE
: The units are out of the range.

MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_INVALID_FLAG	Flag is invalid (*ulParam1*).
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_OUT_OF_MEMORY	Insufficient memory to perform the operation requested.

MCI_DELETE - Remarks

Neither a user-defined buffer nor the clipboard is used by this command. If neither MCI_FROM nor MCI_TO are specified, the range to be deleted is assumed to be from the current position to the end of the file. The difference between MCI_FROM and MCI_TO must be greater than zero, otherwise an error is returned.

The units of the MCI_FROM and MCI_TO parameters are interpreted in the currently selected time format.

The following example illustrates how the MCI_FROM and MCI_TO parameters are interpreted. If a multimedia element is composed of samples and a file has 100 samples; the samples are numbered from 0 to 99. If the from position is specified as 25 and the to position is specified as 30, MCI_DELETE will delete samples 25, 26, 27, 28, and 29. After the delete, the current position of the media would be at sample 25.

Edited Audio/Video Interleaved (AVI) movie files cannot always be saved with their original name after the delete operation. If the clipboard contains a reference to data that would be erased during saving or if another instance of the digital video device has a pending paste operation which depends on this data, the file cannot be saved unless a new file name has been provided. If a new file name is not provided, MMIOERR_NEED_NEW_FILENAME is returned by the AVI I/O procedure and a temporary file is created to save the edited movie.

Note: AVI is the only video file format supporting editing commands.

MCI_DELETE - Related Messages

- MCI_COPY
- MCI_CUT
- MCI_PASTE
- MCI_UNDO
- MCI_REDO

MCI_DELETE - Example Code

The following code illustrates how to delete the first five seconds of a file.

```
    USHORT                usDeviceID;
    MCI_EDIT_PARMS        mep;

    mep.hwndCallback = hwndMyWindow;
    mep.ulFrom = 0;
    mep.ulTo = 5000;   /* Current time format is milliseconds */

                       /* Delete first five seconds of file   */
    mciSendCommand( usDeviceID,
                    MCI_DELETE,
                    MCI_NOTIFY | MCI_FROM | MCI_TO,
                    &mep,
                    0 );
```

MCI_DELETE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_DEVICESETTINGS

MCI_DEVICESETTINGS Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: The MCI_NOTIFY flag is not valid for this message.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_DEVICESETTINGS Parameter - pParam2

pParam2 (PMCI_DEVICESETTINGS_PARMS)
A pointer to the MCI_DEVICESETTINGS_PARMS data structure.

MCI_DEVICESETTINGS Return Value - hwnd

hwnd (HWND)
Returns the handle to a settings page or zero if no page is inserted.

MCI_DEVICESETTINGS - Description

This message is sent to a media control interface driver (MCD) when the Multimedia Setup application is inserting pages into a Settings notebook. This message provides the MCD the opportunity to insert custom settings pages.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: The MCI_NOTIFY flag is not valid for this message.

MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_DEVICESETTINGS_PARMS)
A pointer to the MCI_DEVICESETTINGS_PARMS data structure.

hwnd (HWND)
Returns the handle to a settings page or zero if no page is inserted.

MCI_DEVICESETTINGS - Remarks

This message is sent only if the MCI_SYSINFO_DEVICESETTINGS flag is set in the *ulDeviceFlag* field of the MCI_SYSINFO_LOGDEVICE data structure. Refer to the *OS/2 Multimedia Subsystem Programming Guide* for details of inserting settings pages.

This command does not require the device to be opened.

Note: This command is used mainly by the Multimedia Setup application and should not be used by general purpose OS/2 multimedia applications.

MCI_DEVICESETTINGS - Example Code

The following code illustrates how to close a device context.

```
  ULONG mciDriverEntry (PINSTANCE pInst,
                        USHORT    usMessage,
                        ULONG     ulParam1,
                        ULONG     ulParam2,
                        USHORT    usUserParam)

{
  switch  (usMessage)  {
     case MCI_DEVICESETTINGS:
       return(InsertPage  ((PMCI_DEVICE_SETTINGS_PARMS) ulParam2));
}
}
```

MCI_DEVICESETTINGS - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_ESCAPE

MCI_ESCAPE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ESCAPE_STRING
This flag indicates a command string is specified in the *pszCommand* field of the MCI_ESCAPE_PARMS data structure.

MCI_ESCAPE Parameter - pParam2

pParam2 (PMCI_ESCAPE_PARMS)
A pointer to the MCI_ESCAPE_PARMS data structure.

MCI_ESCAPE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make the device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_ESCAPE - Description

This message sends messages directly to the vendor-specific driver (VSD) or the device driver. This message is not interpreted by the media control interface driver (MCD).

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_ESCAPE_STRING
This flag indicates a command string is specified in the *pszCommand* field of the MCI_ESCAPE_PARMS data structure.

pParam2 (PMCI_ESCAPE_PARMS)
A pointer to the MCI_ESCAPE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INSTANCE_INACTIVE	The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make the device ID active.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_HARDWARE	Device hardware error.
MCIERR_UNSUPPORTED_FUNCTION	Unsupported function.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_MISSING_PARAMETER	Required parameter is missing.

MCI_ESCAPE - Remarks

MCI_ESCAPE provides a means of passing a command string directly to a VSD or device driver for execution.

Support of this message is optional.

MCI_ESCAPE - Topics

Select an item:
Description
Returns
Remarks
Glossary

MCI_FREEZE

MCI_FREEZE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Video Overlay Extensions

The following additional items apply to video overlay devices:

MCI_OVLY_FREEZE_RECT
Specifies that the *rc* field of the MCI_OVLY_RECT_PARMS data structure contains a valid rectangle. If this flag is not specified, the entire image is frozen.

MCI_OVLY_FREEZE_RECT_OUTSIDE
Specifies that the area outside the specified rectangle is to be affected. If this flag is not specified then the area inside is affected. This flag must be specified with the MCI_OVLY_FREEZE_RECT flag.

MCI_FREEZE Parameter - pParam2

pParam2 (PMCI_OVLY_RECT_PARMS)
A pointer to the MCI_OVLY_RECT_PARMS data structure.

MCI_FREEZE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available; for example, because video has been set off.

MCI_FREEZE - Description

This message freezes the motion video on an area of the display.

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Video Overlay Extensions

The following additional items apply to video overlay devices:

MCI_OVLY_FREEZE_RECT
Specifies that the *rc* field of the MCI_OVLY_RECT_PARMS data structure contains a valid rectangle. If this flag is not specified, the entire image is frozen.

MCI_OVLY_FREEZE_RECT_OUTSIDE
Specifies that the area outside the specified rectangle is to be affected. If this flag is not specified then the area inside is affected. This flag must be specified with the MCI_OVLY_FREEZE_RECT flag.

pParam2 (PMCI_OVLY_RECT_PARMS)
A pointer to the MCI_OVLY_RECT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG

Invalid flag specified for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available; for example, because video has been set off.

MCI_FREEZE - Remarks

MCI_FREEZE differs from MCI_PAUSE in that it causes the video overlay device to cease updating the video image without affecting the state of the image source device (external video device). For example, if a motion video is being played and MCI_FREEZE is issued, the motion video continues to play but its display is frozen.

Freezing or unfreezing an area outside the current video destination rectangle has no effect.

Multiple freeze and unfreeze commands, which specify rectangles to be affected, can be issued sequentially to build up a complex region of frozen and unfrozen video.

MCI_FREEZE - Default Processing

If MCI_OVLY_FREEZE_RECT is not specified, the entire image is frozen. If MCI_OVLY_FREEZE_RECT_OUTSIDE is not specified, the default is the area inside the rectangle.

MCI_FREEZE - Related Messages

- MCI_UNFREEZE

MCI_FREEZE - Example Code

The following code illustrates how to freeze the motion of a video image.

```
MCI_VID_RECT_PARMS mciFreezeParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;

/* Freezing OUTSIDE a sub-rectangle of the window */
memset (&mciFreezeParms, 0x00, sizeof (MCI_VID_RECT_PARMS));
mciFreezeParms.hwndCallback = hwndNotify;
mciFreezeParms.rc.xLeft   = lX1;
mciFreezeParms.rc.yBottom = lY1;
mciFreezeParms.rc.xRight  = lX2;
mciFreezeParms.rc.yTop    = lY2;

ulReturn = mciSendCommand(usDeviceID,
                      MCI_FREEZE,
                      MCI_WAIT |
                      MCI_OVLY_FREEZE_RECT_OUTSIDE |
                      MCI_OVLY_FREEZE
                      (PVOID)&mciFreezeParms,
                      usUserParm);
```

MCI_FREEZE - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Related Messages
Example Code
Glossary

MCI_GETDEVCAPS

MCI_GETDEVCAPS Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: Either MCI_GETDEVCAPS_MESSAGE or MCI_GETDEVCAPS_ITEM must be specified.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_GETDEVCAPS_EXTENDED
Indicates extended device capabilities are required. (Specifying MCI_GETDEVCAPS_EXTENDED implies MCI_GETDEVCAPS_ITEM.) See the individual device-specific extensions for each device for use of this flag.

MCI_GETDEVCAPS_MESSAGE
The *usMessage* field of the data structure identified by *pParam2* contains a constant specifying the message to be queried. If the device supports the message, MCI_TRUE is returned; otherwise, MCI_FALSE is returned.

Note: The string parser converts unrecognized strings into a message ID value of 0. This message value is defined as not being supported by any driver. Other messages are converted to their corresponding message ID value.

MCI_GETDEVCAPS_ITEM
The *ulItem* field of the data structure identified by *pParam2* contains a constant specifying the device capabilities to be queried.

The following list of items can be used regardless of the type of device:

MCI_GETDEVCAPS_CAN_EJECT
Returns MCI_TRUE if the device can eject its media; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_LOCKEJECT
Returns MCI_TRUE if the device can disable the manual ejection of its media; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_PLAY
Returns MCI_TRUE if the device can play its media; otherwise, it returns MCI_FALSE. If the device returns MCI_TRUE, the device supports MCI_PLAY, MCI_PAUSE, MCI_RESUME, and MCI_STOP.

MCI_GETDEVCAPS_CAN_PROCESS_INTERNAL
Returns MCI_TRUE if the device can internally process digital data such as a CD Digital Audio drive with a built-in digital-to-analog converter (DAC); otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_RECORD
Returns MCI_TRUE if the device can record its media; otherwise, it returns MCI_FALSE. If MCI_TRUE is returned, the device supports MCI_RECORD.

MCI_GETDEVCAPS_CAN_RECORD_INSERT
Returns MCI_TRUE if the device supports insertion of data while recording; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_SAVE
Returns MCI_TRUE if the device can save files; otherwise, it returns MCI_FALSE. If a device returns TRUE, the MCI_SAVE command must be issued to save changes in the media file.

MCI_GETDEVCAPS_CAN_SETVOLUME
Returns MCI_TRUE if the device can change the audio volume level; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_STREAM
Returns MCI_TRUE if the device can stream digital data continuously to or from memory; otherwise, it returns MCI_FALSE. The source or destination of the data transfer is determined by the device instance connection.

MCI_GETDEVCAPS_DEVICE_TYPE
Returns the constant defined for this particular device type.

MCI_GETDEVCAPS_HAS_AUDIO
Returns MCI_TRUE if the device is capable of playing audio; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_HAS_IMAGE
Returns MCI_TRUE if the device supports a still image in its device instance; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_HAS_VIDEO
Returns MCI_TRUE if the device is capable of playing video; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_PREROLL_TIME
Returns a deterministic or maximum notified preroll time in MMTIME units (regardless of the currently set time base for the device). A value of 0 for the maximum notified preroll time indicates that an upper boundary to the preroll time is not known.

MCI_GETDEVCAPS_PREROLL_TYPE
Returns MCI_PREROLL_NONE.

MCI_GETDEVCAPS_USES_FILES
Returns MCI_TRUE if the device requires a file name or playlist pointer; otherwise, it returns MCI_FALSE.

Amplifier Mixer Extensions

If the MCI_GETDEVCAPS_EXTENDED flag is specified, the following flags can be placed in the *ulAttribute* field of MCI_AMP_GETDEVCAPS_PARMS. The *ulExtended* field of the MCI_AMP_GETDEVCAPS_PARMS structure must contain MCI_MIXER_LINE if the MCI_GETDEVCAPS_EXTENDED flag is specified.

MCI_AMP_CAN_SET_TREBLE
This flag allows an application to determine whether treble settings are supported.

MCI_AMP_CAN_SET_MID

This flag allows an application to determine whether mid settings are supported.

MCI_AMP_CAN_SET_BASS
This flag allows an application to determine whether bass settings are supported.

MCI_AMP_CAN_SET_BALANCE
This flag allows an application to determine whether balance settings are supported.

MCI_AMP_CAN_SET_GAIN
This flag allows an application to determine whether gain settings are supported.

MCI_AMP_CAN_SET_VOLUME
This flag allows an application to determine whether volume settings are supported.

MCI_AMP_CAN_SET_MONITOR
This flag allows an application to determine whether monitor settings are supported.

MCI_AMP_CAN_SET_PITCH
This flag allows an application to determine whether pitch settings are supported.

MCI_AMP_CAN_SET_LOUDNESS
This flag allows an application to determine whether loudness settings are supported.

MCI_AMP_CAN_SET_CROSSOVER
This flag allows an application to determine whether crossover settings are supported.

MCI_AMP_CAN_SET_REVERB
This flag allows an application to determine whether reverb settings are supported.

MCI_AMP_CAN_SET_ALC
This flag allows an application to determine whether auto-level controls are supported.

MCI_AMP_CAN_SET_CHORUS
This flag allows an application to determine whether chorus controls are supported.

MCI_AMP_CAN_SET_CUSTOM1
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_CUSTOM2
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_CUSTOM3
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_MUTE
This flag allows an application to determine whether mute settings are supported.

MCI_AMP_CAN_SET_STEREOENHANCE
This flag allows an application to determine whether stereo enhance settings are supported.

Digital Video Extensions

The following additional items apply to digital video devices:

MCI_DGV_GETDEVCAPS_CAN_DISTORT
Returns MCI_TRUE if the device can distort the image independently in horizontal and vertical dimensions; otherwise, it returns MCI_FALSE. Returns MCI_FALSE for most frame-grabber types of hardware, but some hardware (such as Video Blaster) is capable of performing independent scaling in the horizontal and vertical directions and returns MCI_TRUE.

MCI_DGV_GETDEVCAPS_CAN_REVERSE
Returns MCI_TRUE if the device can play in reverse; otherwise, it returns MCI_FALSE.

MCI_DGV_GETDEVCAPS_CAN_STRETCH
Returns MCI_TRUE if the device can stretch the image to fill the frame; otherwise, it returns MCI_FALSE. Returns MCI_FALSE for most frame-grabber types of hardware, but some hardware (such as Video Blaster) is capable of performing scaling and returns MCI_TRUE.

MCI_DGV_GETDEVCAPS_FAST_RATE
Returns the standard fast playback rate (twice the recorded playback rate) in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play fast.

MCI_DGV_GETDEVCAPS_SLOW_RATE

Returns the standard slow playback rate (half the recorded playback rate) in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the slow playback rate.

MCI_DGV_GETDEVCAPS_NORMAL_RATE
Returns the recorded playback rate in the current speed format, either as a percentage or in frames per second.

MCI_DGV_GETDEVCAPS_VIDEO_X_EXTENT
Returns the nominal horizontal (X) extent of the digital motion video image.

MCI_DGV_GETDEVCAPS_VIDEO_Y_EXTENT
Returns the nominal vertical (Y) extent of the digital motion video image.

MCI_DGV_GETDEVCAPS_IMAGE_X_EXTENT
Returns the nominal horizontal (X) extent of images, if applicable.

MCI_DGV_GETDEVCAPS_IMAGE_Y_EXTENT
Returns the nominal vertical (Y) extent of images, if applicable.

MCI_DGV_GETDEVCAPS_OVERLAY_GRAPHICS
Returns MCI_TRUE if the device supports overlaying video with application-generated graphics, otherwise returns MCI_FALSE. Overlay cards such as Video Blaster enable graphics overlay of the hardware monitor window, however, overlay is not supported over video playback in the graphics buffer.

MCI_DGV_GETDEVCAPS_HAS_TUNER
Returns MCI_TRUE if the device has TV tuner capabilities.

Videodisc Extensions

The following additional item values apply to videodisc devices:

MCI_VD_GETDEVCAPS_CAN_REVERSE
Returns MCI_TRUE if the videodisc player can play in reverse; otherwise, it returns MCI_FALSE. Some players can play CLV discs in reverse as well as CAV discs.

MCI_VD_GETDEVCAPS_FAST_RATE
Returns the standard fast play rate in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the fast play rate.

MCI_VD_GETDEVCAPS_SLOW_RATE
Returns the standard slow play rate in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the slow play rate.

MCI_VD_GETDEVCAPS_NORMAL_RATE
Returns the normal rate of play in frames per second.

MCI_VD_GETDEVCAPS_MAXIMUM_RATE
Returns the maximum play rate in the current speed format, either as a percentage or in frames per second.

MCI_VD_GETDEVCAPS_MINIMUM_RATE
Returns the minimum play rate in the current speed format, either as a percentage or in frames per second. The minimum play rate is the slowest playback rate the device is capable of other than a paused or stopped state, that is, non-zero.

MCI_VD_GETDEVCAPS_CLV
Specifies that the requested capability information is relative to constant linear velocity (CLV) formatted discs.

MCI_VD_GETDEVCAPS_CAV
Specifies that the requested capability information is relative to constant angular velocity (CAV) formatted discs. This is the default.

Video Overlay Extensions

The following additional items apply to video overlay devices:

MCI_OVLY_GETDEVCAPS_CAN_DISTORT
Returns MCI_TRUE if the device can stretch the image independently in horizontal and vertical dimensions; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_CAN_FREEZE
Returns MCI_TRUE if the device can freeze the image; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_CAN_STRETCH

Returns MCI_TRUE if the device can stretch or shrink the image to fill the frame; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_VIDEO_X_EXTENT
Returns the nominal horizontal (X) extent of the video source. Returns 706 for both NTSC and PAL video.

MCI_OVLY_GETDEVCAPS_VIDEO_Y_EXTENT
Returns the nominal vertical (Y) extent of the video source. Returns 484 for NTSC video or 564 for PAL video.

MCI_OVLY_GETDEVCAPS_IMAGE_X_EXTENT
Returns the nominal horizontal (X) extent of images for the device. Returns 640.

MCI_OVLY_GETDEVCAPS_IMAGE_Y_EXTENT
Returns the nominal vertical (Y) extent of images for the device. Returns 480.

MCI_OVLY_GETDEVCAPS_OVERLAY_GRAPHICS
Returns MCI_TRUE if the device supports overlaying video with application-generated graphics; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_MAX_WINDOWS
Returns the maximum number of windows that the device can handle concurrently. Returns 10.

Waveform Audio Extensions

If the MCI_GETDEVCAPS_EXTENDED flag is specified, the following flags can be placed in the *ulItem* field of the MCI_WAVE_GETDEVCAPS_PARMS data structure for the waveaudio device.

MCI_GETDEVCAPS_WAVE_FORMAT
This flag allows an application to determine whether a specific waveaudio format is supported. The application must fill in the *ulBitsPerSample*, *ulFormatTag*, *ulSamplesPerSec*, *ulChannels*, and *ulFormatMode* fields in the MCI_WAVE_GETDEVCAPS_PARMS structure. If the format is supported, the driver returns MCI_TRUE. If the format is not supported, the driver returns a return code that indicates why the command failed.

MCI_GETDEVCAPS Parameter - pParam2

pParam2 (PMCI_GETDEVCAPS_PARMS)
A pointer to the MCI_GETDEVCAPS_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_GETDEVCAPS_PARMS
A pointer to the MCI_AMP_GETDEVCAPS_PARMS structure.

PMCI_WAVE_GETDEVCAPS_PARMS
A pointer to the MCI_WAVE_GETDEVCAPS_PARMS structure.

MCI_GETDEVCAPS Return Value - rc

rc (ULONG)
The low-order word of *rc* contains a code indicating success or failure:

```
/* Only examine the low-order word of the return code for */
/*success/failure */
if ( (ulError & 0x0000FFFF) == MCIERR_SUCCESS )
```

The format of the *ulReturn* value in the MCI_GETDEVCAPS_PARMS structure is defined by the high-order word of the value returned by mciSendCommand. This value is used by mciSendString to determine how to convert the *ulReturn* value to string form. For a list of the possible format values, see the MMDRVOS2.H header file.

Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_FLAGS_NOT_COMPATIBLE	The flags cannot be used together.
MCIERR_INVALID_CONNECTOR_TYPE	Invalid connector type given.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_INVALID_ITEM_FLAG	Invalid item flag specified for this command.
MCIERR_MISSING_FLAG	Flag missing for this MMPM/2 command.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_UNSUPPORTED_CONN_TYPE	Connector type is not supported by this device.
MCIERR_UNSUPP_CHANNELS	The hardware does not support this channel setting.
MCIERR_UNSUPP_BITSPERSAMPLE	The hardware does not support this bits per sample setting.
MCIERR_UNSUPP_FORMAT_MODE	The hardware does not support this format mode.
MCIERR_UNSUPP_FORMAT_TAG	The hardware does not support this format tag.
MCIERR_UNSUPP_SAMPLESPERSEC	The hardware does not support this sampling rate.
MCIERR_UNSUPPORTED_ATTRIBUTE	Current mixer hardware does not support the attribute.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.

MCI_GETDEVCAPS - Description

This message is used to return static information about the capabilities of a particular device instance.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: Either MCI_GETDEVCAPS_MESSAGE or MCI_GETDEVCAPS_ITEM must be specified.

MCI_NOTIFY

A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT

Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_GETDEVCAPS_EXTENDED

Indicates extended device capabilities are required. (Specifying MCI_GETDEVCAPS_EXTENDED implies MCI_GETDEVCAPS_ITEM.) See the individual device-specific extensions for each device for use of this flag.

MCI_GETDEVCAPS_MESSAGE

The *usMessage* field of the data structure identified by *pParam2* contains a constant specifying the message to be queried. If the device supports the message, MCI_TRUE is returned; otherwise, MCI_FALSE is returned.

Note: The string parser converts unrecognized strings into a message ID value of 0. This message value is defined as not being supported by any driver. Other messages are converted to their corresponding message ID value.

MCI_GETDEVCAPS_ITEM

The *ulItem* field of the data structure identified by *pParam2* contains a constant specifying the device capabilities to be queried.

The following list of items can be used regardless of the type of device:

MCI_GETDEVCAPS_CAN_EJECT

Returns MCI_TRUE if the device can eject its media; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_LOCKEJECT

Returns MCI_TRUE if the device can disable the manual ejection of its media; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_PLAY

Returns MCI_TRUE if the device can play its media; otherwise, it returns MCI_FALSE. If the device returns MCI_TRUE, the device supports MCI_PLAY, MCI_PAUSE, MCI_RESUME, and MCI_STOP.

MCI_GETDEVCAPS_CAN_PROCESS_INTERNAL

Returns MCI_TRUE if the device can internally process digital data such as a CD Digital Audio drive with a built-in digital-to-analog converter (DAC); otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_RECORD

Returns MCI_TRUE if the device can record its media; otherwise, it returns MCI_FALSE. If MCI_TRUE is returned, the device supports MCI_RECORD.

MCI_GETDEVCAPS_CAN_RECORD_INSERT

Returns MCI_TRUE if the device supports insertion of data while recording; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_SAVE

Returns MCI_TRUE if the device can save files; otherwise, it returns MCI_FALSE. If a device returns TRUE, the MCI_SAVE command must be issued to save changes in the media file.

MCI_GETDEVCAPS_CAN_SETVOLUME

Returns MCI_TRUE if the device can change the audio volume level; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_CAN_STREAM

Returns MCI_TRUE if the device can stream digital data continuously to or from memory; otherwise, it returns MCI_FALSE. The source or destination of the data transfer is determined by the device instance connection.

MCI_GETDEVCAPS_DEVICE_TYPE

Returns the constant defined for this particular device type.

MCI_GETDEVCAPS_HAS_AUDIO

Returns MCI_TRUE if the device is capable of playing audio; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_HAS_IMAGE

Returns MCI_TRUE if the device supports a still image in its device instance; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_HAS_VIDEO
Returns MCI_TRUE if the device is capable of playing video; otherwise, it returns MCI_FALSE.

MCI_GETDEVCAPS_PREROLL_TIME
Returns a deterministic or maximum notified preroll time in MMTIME units (regardless of the currently set time base for the device). A value of 0 for the maximum notified preroll time indicates that an upper boundary to the preroll time is not known.

MCI_GETDEVCAPS_PREROLL_TYPE
Returns MCI_PREROLL_NONE.

MCI_GETDEVCAPS_USES_FILES
Returns MCI_TRUE if the device requires a file name or playlist pointer; otherwise, it returns MCI_FALSE.

Amplifier Mixer Extensions

If the MCI_GETDEVCAPS_EXTENDED flag is specified, the following flags can be placed in the *ulAttribute* field of MCI_AMP_GETDEVCAPS_PARMS. The *ulExtended* field of the MCI_AMP_GETDEVCAPS_PARMS structure must contain MCI_MIXER_LINE if the MCI_GETDEVCAPS_EXTENDED flag is specified.

MCI_AMP_CAN_SET_TREBLE
This flag allows an application to determine whether treble settings are supported.

MCI_AMP_CAN_SET_MID
This flag allows an application to determine whether mid settings are supported.

MCI_AMP_CAN_SET_BASS
This flag allows an application to determine whether bass settings are supported.

MCI_AMP_CAN_SET_BALANCE
This flag allows an application to determine whether balance settings are supported.

MCI_AMP_CAN_SET_GAIN
This flag allows an application to determine whether gain settings are supported.

MCI_AMP_CAN_SET_VOLUME
This flag allows an application to determine whether volume settings are supported.

MCI_AMP_CAN_SET_MONITOR
This flag allows an application to determine whether monitor settings are supported.

MCI_AMP_CAN_SET_PITCH
This flag allows an application to determine whether pitch settings are supported.

MCI_AMP_CAN_SET_LOUDNESS
This flag allows an application to determine whether loudness settings are supported.

MCI_AMP_CAN_SET_CROSSOVER
This flag allows an application to determine whether crossover settings are supported.

MCI_AMP_CAN_SET_REVERB
This flag allows an application to determine whether reverb settings are supported.

MCI_AMP_CAN_SET_ALC
This flag allows an application to determine whether auto-level controls are supported.

MCI_AMP_CAN_SET_CHORUS
This flag allows an application to determine whether chorus controls are supported.

MCI_AMP_CAN_SET_CUSTOM1
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_CUSTOM2
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_CUSTOM3
This flag allows an application to determine whether a custom effect is supported.

MCI_AMP_CAN_SET_MUTE

This flag allows an application to determine whether mute settings are supported.

MCI_AMP_CAN_SET_STEREOENHANCE
This flag allows an application to determine whether stereo enhance settings are supported.

Digital Video Extensions

The following additional items apply to digital video devices:

MCI_DGV_GETDEVCAPS_CAN_DISTORT
Returns MCI_TRUE if the device can distort the image independently in horizontal and vertical dimensions; otherwise, it returns MCI_FALSE. Returns MCI_FALSE for most frame-grabber types of hardware, but some hardware (such as Video Blaster) is capable of performing independent scaling in the horizontal and vertical directions and returns MCI_TRUE.

MCI_DGV_GETDEVCAPS_CAN_REVERSE
Returns MCI_TRUE if the device can play in reverse; otherwise, it returns MCI_FALSE.

MCI_DGV_GETDEVCAPS_CAN_STRETCH
Returns MCI_TRUE if the device can stretch the image to fill the frame; otherwise, it returns MCI_FALSE. Returns MCI_FALSE for most frame-grabber types of hardware, but some hardware (such as Video Blaster) is capable of performing scaling and returns MCI_TRUE.

MCI_DGV_GETDEVCAPS_FAST_RATE
Returns the standard fast playback rate (twice the recorded playback rate) in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play fast.

MCI_DGV_GETDEVCAPS_SLOW_RATE
Returns the standard slow playback rate (half the recorded playback rate) in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the slow playback rate.

MCI_DGV_GETDEVCAPS_NORMAL_RATE
Returns the recorded playback rate in the current speed format, either as a percentage or in frames per second.

MCI_DGV_GETDEVCAPS_VIDEO_X_EXTENT
Returns the nominal horizontal (X) extent of the digital motion video image.

MCI_DGV_GETDEVCAPS_VIDEO_Y_EXTENT
Returns the nominal vertical (Y) extent of the digital motion video image.

MCI_DGV_GETDEVCAPS_IMAGE_X_EXTENT
Returns the nominal horizontal (X) extent of images, if applicable.

MCI_DGV_GETDEVCAPS_IMAGE_Y_EXTENT
Returns the nominal vertical (Y) extent of images, if applicable.

MCI_DGV_GETDEVCAPS_OVERLAY_GRAPHICS
Returns MCI_TRUE if the device supports overlaying video with application-generated graphics, otherwise returns MCI_FALSE. Overlay cards such as Video Blaster enable graphics overlay of the hardware monitor window, however, overlay is not supported over video playback in the graphics buffer.

MCI_DGV_GETDEVCAPS_HAS_TUNER
Returns MCI_TRUE if the device has TV tuner capabilities.

Videodisc Extensions

The following additional item values apply to videodisc devices:

MCI_VD_GETDEVCAPS_CAN_REVERSE
Returns MCI_TRUE if the videodisc player can play in reverse; otherwise, it returns MCI_FALSE. Some players can play CLV discs in reverse as well as CAV discs.

MCI_VD_GETDEVCAPS_FAST_RATE
Returns the standard fast play rate in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the fast play rate.

MCI_VD_GETDEVCAPS_SLOW_RATE
Returns the standard slow play rate in the current speed format, either as a percentage or in frames per second. Returns the normal play rate if the device cannot play at the slow play rate.

MCI_VD_GETDEVCAPS_NORMAL_RATE
Returns the normal rate of play in frames per second.

MCI_VD_GETDEVCAPS_MAXIMUM_RATE
Returns the maximum play rate in the current speed format, either as a percentage or in frames per second.

MCI_VD_GETDEVCAPS_MINIMUM_RATE
Returns the minimum play rate in the current speed format, either as a percentage or in frames per second. The minimum play rate is the slowest playback rate the device is capable of other than a paused or stopped state, that is, non-zero.

MCI_VD_GETDEVCAPS_CLV
Specifies that the requested capability information is relative to constant linear velocity (CLV) formatted discs.

MCI_VD_GETDEVCAPS_CAV
Specifies that the requested capability information is relative to constant angular velocity (CAV) formatted discs. This is the default.

Video Overlay Extensions

The following additional items apply to video overlay devices:

MCI_OVLY_GETDEVCAPS_CAN_DISTORT
Returns MCI_TRUE if the device can stretch the image independently in horizontal and vertical dimensions; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_CAN_FREEZE
Returns MCI_TRUE if the device can freeze the image; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_CAN_STRETCH
Returns MCI_TRUE if the device can stretch or shrink the image to fill the frame; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_VIDEO_X_EXTENT
Returns the nominal horizontal (X) extent of the video source. Returns 706 for both NTSC and PAL video.

MCI_OVLY_GETDEVCAPS_VIDEO_Y_EXTENT
Returns the nominal vertical (Y) extent of the video source. Returns 484 for NTSC video or 564 for PAL video.

MCI_OVLY_GETDEVCAPS_IMAGE_X_EXTENT
Returns the nominal horizontal (X) extent of images for the device. Returns 640.

MCI_OVLY_GETDEVCAPS_IMAGE_Y_EXTENT
Returns the nominal vertical (Y) extent of images for the device. Returns 480.

MCI_OVLY_GETDEVCAPS_OVERLAY_GRAPHICS
Returns MCI_TRUE if the device supports overlaying video with application-generated graphics; otherwise, it returns MCI_FALSE.

MCI_OVLY_GETDEVCAPS_MAX_WINDOWS
Returns the maximum number of windows that the device can handle concurrently. Returns 10.

Waveform Audio Extensions

If the MCI_GETDEVCAPS_EXTENDED flag is specified, the following flags can be placed in the *ulItem* field of the MCI_WAVE_GETDEVCAPS_PARMS data structure for the waveaudio device.

MCI_GETDEVCAPS_WAVE_FORMAT
This flag allows an application to determine whether a specific waveaudio format is supported. The application must fill in the *ulBitsPerSample* , *ulFormatTag* , *ulSamplesPerSec* , *ulChannels* , and *ulFormatMode* fields in the MCI_WAVE_GETDEVCAPS_PARMS structure. If the format is supported, the driver returns MCI_TRUE. If the format is not supported, the driver returns a return code that indicates why the command failed.

pParam2 (PMCI_GETDEVCAPS_PARMS)
A pointer to the MCI_GETDEVCAPS_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_GETDEVCAPS_PARMS
A pointer to the MCI_AMP_GETDEVCAPS_PARMS structure.

PMCI_WAVE_GETDEVCAPS_PARMS
A pointer to the MCI_WAVE_GETDEVCAPS_PARMS structure.

rc (ULONG)
The low-order word of *rc* contains a code indicating success or failure:

```
/* Only examine the low-order word of the return code for */
/*success/failure */
if ( (ulError & 0x0000FFFF) == MCIERR_SUCCESS )
```

The format of the *ulReturn* value in the MCI_GETDEVCAPS_PARMS structure is defined by the high-order word of the value returned by mciSendCommand. This value is used by mciSendString to determine how to convert the *ulReturn* value to string form. For a list of the possible format values, see the MMDRVOS2.H header file.

Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_FLAGS_NOT_COMPATIBLE	The flags cannot be used together.
MCIERR_INVALID_CONNECTOR_TYPE	Invalid connector type given.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_INVALID_ITEM_FLAG	Invalid item flag specified for this command.
MCIERR_MISSING_FLAG	Flag missing for this MMPM/2 command.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_UNSUPPORTED_CONN_TYPE	Connector type is not supported by this device.
MCIERR_UNSUPP_CHANNELS	The hardware does not support this channel setting.
MCIERR_UNSUPP_BITSPERSAMPLE	The hardware does not support this bits per sample setting.
MCIERR_UNSUPP_FORMAT_MODE	The hardware does not support this format mode.
MCIERR_UNSUPP_FORMAT_TAG	The hardware does not support this format tag.
MCIERR_UNSUPP_SAMPLESPERSEC	The hardware does not support this sampling rate.
MCIERR_UNSUPPORTED_ATTRIBUTE	Current mixer hardware does not support the attribute.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.

MCI_GETDEVCAPS - Remarks

The MCI_GETDEVCAPS_ITEM and MCI_GETDEVCAPS_MESSAGE flags are mutually exclusive. Only a single item or message can be specified.

MCI_DGV_GETDEVCAPS_MINIMUM_RATE, MCI_DGV_GETDEVCAPS_MAXIMUM_RATE, and MCI_DGV_GETDEVCAPS_MAX_WINDOWS are not supported. If these flags are specified, MCIERR_UNSUPPORTED_FLAG is returned.

MCI_DGV_GETDEVCAPS_VIDEO_X_EXTENT, MCI_DGV_GETDEVCAPS_VIDEO_Y_EXTENT, MCI_DGV_GETDEVCAPS_IMAGE_X_EXTENT, and MCI_DGV_GETDEVCAPS_IMAGE_Y_EXTENT return hardware-specific values from the vendor-specific driver (VSD). This is normally the size of the video capture card's frame buffer.

The values for video extent specify the largest video image that can be captured and thereby define the extents of the video capture coordinate system. Capture regions specified by MCI_PUT must lie entirely within these extents.

The values for image extent specify the largest still image that can be captured with the device. The values returned are the same as video extents for supported hardware.

MCI_GETDEVCAPS - Default Processing

For videodisc devices, the MCI_VD_GETDEVCAPS_CAV flag is the default.

MCI_GETDEVCAPS - Example Code

The following code illustrates how to determine if a device has audio capability.

```
   USHORT   usDeviceID;
   ULONG    rc;
   BOOL     fHas_audio;                  /* Set to TRUE by this example
                                            if device has audio      */
   MCI_GETDEVCAPS_PARMS  mgdcp;

   /* Determine if device has audio capability */

   mgdcp.ulItem = MCI_GETDEVCAPS_HAS_AUDIO;

   rc = mciSendCommand(usDeviceID,           /* Device ID                */
                       MCI_GETDEVCAPS,       /* Get device capability
                                                message                  */
                       MCI_WAIT | MCI_GETDEVCAPS_ITEM,
                                             /* Flags for this message */
                       (PVOID) &mgdcp,       /* Data structure           */
                       0);                   /* No user parm             */

   if (LOUSHORT(rc) == MCIERR_SUCCESS)
     {
      fHas_audio = (BOOL) mgdcp.ulReturn; /* Return if device
                                                has audio                */
     }
```

The following example illustrates how an application can determine if it can set the volume attribute for a particular connector.

```
    ULONG  rc;
    MCI_AMP_GETDEVCAPS_PARMS mciAmpCaps;
    USHORT usDeviceID;

/* Test to see if the mixer supports volume changes on the mic. */
    mciAmpCaps.ulValue = MCI_MICROPHONE_CONNECTOR;
    mciAmpCaps.ulAttribute = MCI_AMP_CAN_SET_VOLUME;
    mciAmpCaps.ulExtended = MCI_MIXER_LINE;
    rc = mciSendCommand(usDeviceID,
                        MCI_GETDEVCAPS,
                        MCI_WAIT |
                        MCI_GETDEVCAPS_EXTENDED,
                        (ULONG)&mciAmpCaps,
                        0);
```

MCI_GETDEVCAPS - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Example Code
Glossary

MCI_GETIMAGEBUFFER

MCI_GETIMAGEBUFFER Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CONVERT
Specifies that the image format will be converted to the OS/2 bitmap format. The default is the device-specific format.

Digital Video Extensions

The following flag applies to digital video devices:

MCI_USE_HW_BUFFER
If this flag is specified, a capture will be from the capture video buffer. If this flag is *not* specified, a capture will be from the movie element and not the contents of the capture video buffer generated by MCI_CAPTURE.

Video Overlay Extensions

The following flag applies to video overlay devices:

MCI_GET_HW_BUFFER_PTR
Requests a pointer to the hardware buffer.

M-Motion specific: Not supported.

MCI_USE_HW_BUFFER
Indicates that the hardware buffer contains the image data.

MCI_GETIMAGEBUFFER Parameter - pParam2

pParam2 (PMCI_IMAGE_PARMS)
A pointer to the MCI_IMAGE_PARMS data structure. If the *pPelBuffer* field in this data structure is 0, this command is treated as a query, and the other fields in the structure are filled in by the driver.

MCI_GETIMAGEBUFFER Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported by this device.
MCIERR_INSTANCE_INACTIVE	Instance inactive.
MCIERR_INVALID_BUFFER	Invalid return buffer given or buffer too small.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_TARGET_DEVICE_FULL	Target device is full.

MCI_GETIMAGEBUFFER - Description

The digital video device uses this message to retrieve the contents of the capture video buffer *or* the current movie frame. (See MCI_CAPTURE for capturing the current movie frame without providing an application buffer.)

Note: Video overlay devices can use this message to read the data in the element buffer that was captured with the MCI_CAPTURE command, obtained by the MCI_LOAD command, or provided by the MCI_SETIMAGEBUFFER command.

The image data is returned in the device-specific format, unless MCI_CONVERT is specified, in which case the data is returned in OS/2 memory bitmap format. The current values for PELFORMAT and BITSPERPEL will be used if possible. The data will be uncompressed.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CONVERT
Specifies that the image format will be converted to the OS/2 bitmap format. The default is the device-specific format.

Digital Video Extensions

The following flag applies to digital video devices:

MCI_USE_HW_BUFFER
If this flag is specified, a capture will be from the capture video buffer. If this flag is *not* specified, a capture will be from the movie element and not the contents of the capture video buffer generated by MCI_CAPTURE.

Video Overlay Extensions

The following flag applies to video overlay devices:

MCI_GET_HW_BUFFER_PTR
Requests a pointer to the hardware buffer.

M-Motion specific: Not supported.

MCI_USE_HW_BUFFER
Indicates that the hardware buffer contains the image data.

pParam2 (PMCI_IMAGE_PARMS)
A pointer to the MCI_IMAGE_PARMS data structure. If the *pPelBuffer* field in this data structure is 0, this command is treated as a query, and the other fields in the structure are filled in by the driver.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported by this device.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_INVALID_BUFFER
Invalid return buffer given or buffer too small.

MCIERR_FILE_NOT_FOUND
File not found.

MCIERR_TARGET_DEVICE_FULL
Target device is full.

MCI_GETIMAGEBUFFER - Remarks

This command might not be supported by the digital video device. To determine whether the device supports the command, issue an MCI_GETDEVCAPS query.

The format of the image data returned is specified by the *ulPelFormat* and *usBitCount* fields of the MCI_IMAGE_PARMS data structure (if possible), unless MCI_CONVERT is specified, in which case the data is returned in OS/2 memory bitmap format. The beginning of the buffer contains the BITMAPINFOHEADER2 data, followed by the palette (if any) and the pel data.

On dual-plane image capture hardware devices, the image layer content is assumed. Only visible data can be captured with some hardware, particularly single-plane devices. The image data returned will be uncompressed, in either OS/2 memory bitmap format or device-specific format, based on the setting of the MCI_CONVERT flag.

The current settings for IMAGE BITSPERPEL and IMAGE PELFORMAT will be used if supported by the device. The IMAGE FILEFORMAT and IMAGE COMPRESSION settings will be ignored.

Conversion from internal YUVB format to OS/2 bitmap format is accomplished with an I/O procedure which can use disk space for temporary storage. Therefore, it is possible that errors such as MCIERR_TARGET_DEVICE_FULL (no disk space) can occur.

MCI_GETIMAGEBUFFER - Related Messages

- MCI_CAPTURE
- MCI_SETIMAGEBUFFER
- MCI_SETIMAGEPALETTE

MCI_GETIMAGEBUFFER - Example Code

The following example shows how to capture a bitmap from video.

```
USHORT usUserParm = 0;
BITMAPINFOHEADER2 *pBMPhdr;
ULONG   ulReturn;
CHAR    szInfoStr[500];
CHAR    szTempStr[100];
ULONG   ulFlags = 0;

   ulFlags = MCI_CONVERT;

/***********************************************************/
/* Determine the length and characteristics of the buffer */
/***********************************************************/
memset ((PVOID)&mciImageParms, 0x00, sizeof (MCI_IMAGE_PARMS));
mciImageParms.hwndCallback = hwndNotify;
mciImageParms.ulBufLen = 0;
mciImageParms.pPelBuffer = 0;

ulReturn = mciSendCommand(usDeviceID, MCI_GETIMAGEBUFFER,
               MCI_WAIT | ulFlags,
               (PVOID)&mciImageParms,
               usUserParm);
/*************************************/
/* Allocate memory for the buffer    */
/*************************************/
```

```
   DosAllocMem (&mciImageParms.pPelBuffer,
                mciImageParms.ulBufLen,
                PAG_COMMIT | PAG_WRITE);

   /*********************************/
   /* Get the data from the buffer  */
   /*********************************/
   ulReturn = mciSendCommand(usDeviceID, MCI_GETIMAGEBUFFER,
                  MCI_WAIT | ulFlags,
                  (PVOID)&mciImageParms,
                  usUserParm);

   pBMPhdr = (BITMAPINFOHEADER2 *)mciImageParms.pPelBuffer;
```

Note: The digital video device returns BITMAPFILEHEADER2 instead of BITMAPINFOHEADER2.

The following code illustrates how to capture an OS/2 bitmap from the hardware using the digital video device.

```
#define INCL_GPI
#define INCL_GPIBITMAPS

#include <os2.h>
#include <pmbitmap.h>

#define  INCL_MMIO
#define  INCL_MMIO_CODEC
#define  INCL_MMIO_DOSIOPROC
#include <os2me.h>
#include <stdlib.h>

/*****************************************************************
 * Name : BMPCaptureBitmap
 *
 * Function: Capture bitmap from hardware
 *
 ******************************************************************/
 VOID BMPCaptureBitmap(PSWVRCB pCB, HWND hwnd)
 {
   MCI_IMAGE_PARMS  mciImageParms;
   PCHAR            pBuf=0L;
   HFILE            hBMP;
   ULONG            ulAction;
   ULONG            cBytes;
   LONG             rc;

   memset ((PVOID)&mciImageParms, 0x00, sizeof (MCI_IMAGE_PARMS));

   /* prepare structures */
   mciImageParms.pPelBuffer   = 0L;
   mciImageParms.ulBufLen     = 0L;

   mciImageParms.rect.xLeft   = pCB->recopts[usIndex].usCapPosX;
   mciImageParms.rect.yBottom = pCB->recopts[usIndex].usCapPosY;
   mciImageParms.rect.xRight  = pCB->recopts[usIndex].usCapSizeX +
                                pCB->recopts[usIndex].usCapPosX;
   mciImageParms.rect.yTop    = pCB->recopts[usIndex]usCapSizeY +
                                pCB->recopts[usIndex].usCapPosY;
   mciImageParms.ulPelBufferWidth  = pCB->recopts[usIndex].usMovieSizeX;
   mciImageParms.ulPelBufferHeight = pCB->recopts[usIndex].usMovieSizeY;

   rc = mciSendCommand( pCB->OutputMovie.usDeviceID,
                        MCI_GETIMAGEBUFFER,
                        MCI_WAIT | MCI_USE_HW_BUFFER | MCI_CONVERT,
                        (ULONG)&mciImageParms,
                        0);

   rc = DosAllocMem ( (PPVOID) &pBuf,
                      (ULONG)   mciImageParms.ulBufLen,
                      (ULONG)   PAG_COMMIT | PAG_READ | PAG_WRITE);
                      mciImageParms.pPelBuffer=(PVOID)pBuf;

   rc = mciSendCommand( pCB->OutputMovie.usDeviceID,
                        MCI_GETIMAGEBUFFER,
                        MCI_WAIT | MCI_USE_HW_BUFFER | MCI_CONVERT,
                        (ULONG)&mciImageParms,
                        0);
   if (!rc)
```

```
    {
     /* getimage buffer is successful open file and write out bitmap */
     rc = DosOpen ( (PSZ)pCB->szBitmapFilename, &hBMP, &ulAction, 0, FILE_NORMAL,
                    FILE_CREATE,
                    OPEN_ACCESS_WRITEONLY | OPEN_SHARE_DENYWRITE,
                    0L);

     rc = DosWrite (hBMP, (PVOID)pBuf,
                    mciImageParms.ulBufLen,
                    &cBytes);

     rc = DosClose (hBMP);
    }

    /* free buffers */
     DosFreeMem( pBuf );

 }
```

MCI_GETIMAGEBUFFER - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_GETIMAGEPALETTE

MCI_GETIMAGEPALETTE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FIND_BEST_REGISTERED
Select the best palette from the registered color maps and return its ID in the *usRegisteredMap* field of the MCI_PALETTE_PARMS data structure.

MCI_QUERY_REGISTERED_MAP
This flag specifies that the palette specified in the *usRegisteredMap* field is to be returned in the array specified in the *pPalette* field. The size of the palette is returned in the *ulPalEntries* field.

MCI_QUERY_REGISTERED_MAP_SIZE
This flag specifies that the size of the palette specified in the *usRegisteredMap* field is to be returned in the *ulPalEntries* field. This can be used to determine the size of the array to use for MCI_QUERY_REGISTERED_MAP.

MCI_GETIMAGEPALETTE Parameter - ulParam2

ulParam2 (PMCI_PALETTE_PARMS)
A pointer to the MCI_PALETTE_PARMS data structure.

MCI_GETIMAGEPALETTE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
The MMPM/2 command completed successfully.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_CALLBACK_HANDLE
The callback handle given is not correct.

MCI_GETIMAGEPALETTE - Description

This message returns the current palette or color map for the currently captured image, if one is available.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FIND_BEST_REGISTERED
: Select the best palette from the registered color maps and return its ID in the *usRegisteredMap* field of the MCI_PALETTE_PARMS data structure.

MCI_QUERY_REGISTERED_MAP
: This flag specifies that the palette specified in the *usRegisteredMap* field is to be returned in the array specified in the *pPalette* field. The size of the palette is returned in the *ulPalEntries* field.

MCI_QUERY_REGISTERED_MAP_SIZE
: This flag specifies that the size of the palette specified in the *usRegisteredMap* field is to be returned in the *ulPalEntries* field. This can be used to determine the size of the array to use for MCI_QUERY_REGISTERED_MAP.

ulParam2 (PMCI_PALETTE_PARMS)
A pointer to the MCI_PALETTE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: The MMPM/2 command completed successfully.

MCIERR_INVALID_DEVICE_ID
: The device ID is not valid.

MCIERR_DEVICE_LOCKED
: The device is acquired for exclusive use.

MCIERR_INVALID_FLAG
: Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCIERR_INVALID_CALLBACK_HANDLE
: The callback handle given is not correct.

MCI_GETIMAGEPALETTE - Remarks

This command might not be supported by the digital video device. To determine whether the device supports the command, issue MCI_GETDEVCAPS.

On dual-layer image capture hardware devices, the image layer content is assumed. The computation of the palette is based only on visible data on some hardware, particularly single-plane devices.

MCI_GETIMAGEPALETTE - Related Messages

- MCI_GETIMAGEBUFFER
- MCI_SETIMAGEPALETTE

MCI_GETIMAGEPALETTE - Topics

Select an item:
Description

Returns
Remarks
Related Messages
Glossary

MCI_GETTOC

MCI_GETTOC Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_GETTOC Parameter - pParam2

pParam2 (PMCI_TOC_PARMS)
A pointer to the MCI_TOC_PARMS data structure.

MCI_GETTOC Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INSTANCE_INACTIVE	The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_MISSING_PARAMETER
Required parameter missing.

MCIERR_DEVICE_NOT_READY
The device is not ready or is being used by another process.

MCI_GETTOC - Description

This message returns a table of contents structure for the currently loaded compact disc.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_TOC_PARMS)
A pointer to the MCI_TOC_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE

Flags cannot be used together.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_MISSING_PARAMETER
Required parameter missing.

MCIERR_DEVICE_NOT_READY
The device is not ready or is being used by another process.

MCI_GETTOC - Remarks

Device and table of contents structure for the currently loaded disc is returned in the MCI_TOC_REC data structure. From this point, the controlling program can select the CD audio object (audio track in this case) to play. If the size of the buffer passed in is too small to hold all the data returned, then the *ulBufSize* field of the MCI_TOC_PARMS structure contains the required buffer size, the error code MCIERR_INVALID_BUFFER is returned, and the buffer contains only as much of the GETTOC data as its size permits.

Note: Not all CD-ROM drives capable of playing digital-audio compact discs support this feature.

MCI_GETTOC - Default Processing

None

MCI_GETTOC - Example Code

The following code illustrates how to get a table of contents structure for the currently loaded device.

```
USHORT   usDeviceID;
MCI_TOC_PARMS tocparms;
#define MAXTOCRECS 30       /* Query up to 30 toc entries           */
MCI_TOC_REC tocrecs[MAXTOCRECS];

/* Get the table of contents for the currently loaded disc          */

tocparms.pBuf = tocrecs;
tocparms.ulBufSize = sizeof(tocrecs);

mciSendCommand(usDeviceID,  /* Device ID                            */
 MCI_GETTOC,                /* Get table of contents message        */
 MCI_WAIT,                  /* Flag for this message                */
 (PVOID) &tocparms,         /* Data structure                       */
 0);                        /* No user parm                         */
```

MCI_GETTOC - Topics

Select an item:

MCI_GROUP

MCI_GROUP Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Note: The MCI_GROUP message supports several flags, some of which can be used in combination with each other. Valid combinations are described in the flag descriptions below. An invalid combination results in the MCIERR_FLAGS_NOT_COMPATIBLE error return code.

MCI_GROUP_MAKE
This flag specifies the creation of a group. MCI_GROUP_MAKE ties several instances together such that a single command sent to the group is actually sent to each instance in the group. This flag can be combined with any of the other group flags except MCI_GROUP_DELETE, in which case an MCIERR_FLAGS_NOT_COMPATIBLE error is returned. Instances must have been previously opened but these instances can be in any mode (such as playing, stopped, paused, and so forth) for this message to be successful. An array of device IDs is provided by the application in the *paulDeviceID* field of the MCI_GROUP_PARMS data structure. The number of these IDs is provided by the application in the *ulNumDevices* field. If one or more device IDs are invalid, then the MCIERR_INVALID_DEVICE_ID error is returned.

If a device ID or alias references an instance already in another group, the MCIERR_ID_ALREADY_IN_GROUP error message is returned.

MCI_GROUP_DELETE
This flag deletes an existing group by disassociating the instances from each other. None of the device instances in the group are closed just the group reference. None of the other flags can be combined with MCI_GROUP_DELETE since the only information required by this flag is a group ID. If any other flags are specified, an MCIERR_FLAGS_NOT_COMPATIBLE error is returned. The MCIERR_INVALID_GROUP_ID error is returned if an invalid ID is passed.

MCI_GROUP_ALIAS
This flag specifies that the *pszGroupAlias* field contains an alias for the group. This flag is valid only with the MCI_GROUP_MAKE flag. The given alias can then be used to refer to the group from the mciSendString interface. If the alias is already in use, the MCIERR_DUPLICATE_ALIAS error is returned.

MCI_GROUP_NOPIECEMEAL
This flag specifies that the group is to be treated as a whole entity rather than a group of separate parts. If one of the parts (instances) becomes inactive, then all the instances in the group become inactive. This flag is only valid with the MCI_GROUP_MAKE flag. If a group is created with the MCI_GROUP_NOPIECEMEAL flag specified and one or more of the device instances is already inactive, then the entire group (all device instances) will be made inactive.

MCI_GROUP Parameter - pParam2

pParam2 (PMCI_GROUP_PARMS)
A pointer to the MCI_GROUP_PARMS structure.

MCI_GROUP Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_DUPLICATE_ALIAS	An alias is already in use.
MCIERR_GROUP_COMMAND	An unsupported GROUP command is sent to a group.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_ID_ALREADY_IN_GROUP	A device ID or alias references an instance already in another group.
MCIERR_INVALID_GROUP_ID	An invalid group ID is passed.

MCI_GROUP - Description

This message allows applications to create and delete groups of device instances. Group commands allow applications to control several devices using a single command.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY

A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT

Control is not to be returned until the action indicated by this message is completed or an error occurs.

Note: The MCI_GROUP message supports several flags, some of which can be used in combination with each other. Valid combinations are described in the flag descriptions below. An invalid combination results in the MCIERR_FLAGS_NOT_COMPATIBLE error return code.

MCI_GROUP_MAKE

This flag specifies the creation of a group. MCI_GROUP_MAKE ties several instances together such that a single command sent to the group is actually sent to each instance in the group. This flag can be combined with any of the other group flags except MCI_GROUP_DELETE, in which case an MCIERR_FLAGS_NOT_COMPATIBLE error is returned. Instances must have been previously opened but these instances can be in any mode (such as playing, stopped, paused, and so forth) for this message to be successful. An array of device IDs is provided by the application in the *paulDeviceID* field of the MCI_GROUP_PARMS data structure. The number of these IDs is provided by the application in the *ulNumDevices* field. If one or more device IDs are invalid, then the MCIERR_INVALID_DEVICE_ID error is returned.

If a device ID or alias references an instance already in another group, the MCIERR_ID_ALREADY_IN_GROUP error message is returned.

MCI_GROUP_DELETE

This flag deletes an existing group by disassociating the instances from each other. None of the device instances in the group are closed just the group reference. None of the other flags can be combined with MCI_GROUP_DELETE since the only information required by this flag is a group ID. If any other flags are specified, an MCIERR_FLAGS_NOT_COMPATIBLE error is returned. The MCIERR_INVALID_GROUP_ID error is returned if an invalid ID is passed.

MCI_GROUP_ALIAS

This flag specifies that the *pszGroupAlias* field contains an alias for the group. This flag is valid only with the MCI_GROUP_MAKE flag. The given alias can then be used to refer to the group from the mciSendString interface. If the alias is already in use, the MCIERR_DUPLICATE_ALIAS error is returned.

MCI_GROUP_NOPIECEMEAL

This flag specifies that the group is to be treated as a whole entity rather than a group of separate parts. If one of the parts (instances) becomes inactive, then all the instances in the group become inactive. This flag is only valid with the MCI_GROUP_MAKE flag. If a group is created with the MCI_GROUP_NOPIECEMEAL flag specified and one or more of the device instances is already inactive, then the entire group (all device instances) will be made inactive.

pParam2 (PMCI_GROUP_PARMS)

A pointer to the MCI_GROUP_PARMS structure.

rc (ULONG)

Return codes indicating success or type of failure:

MCIERR_SUCCESS

If the function succeeds, 0 is returned.

MCIERR_DUPLICATE_ALIAS

An alias is already in use.

MCIERR_GROUP_COMMAND

An unsupported GROUP command is sent to a group.

MCIERR_FLAGS_NOT_COMPATIBLE

Flags cannot be used together.

MCIERR_ID_ALREADY_IN_GROUP

A device ID or alias references an instance already in another group.

MCIERR_INVALID_GROUP_ID

An invalid group ID is passed.

MCI_GROUP - Remarks

Once a group is created, certain messages sent to the group's ID (or alias) are in turn sent to each device making up that group. The following messages can be sent to a group.

MCI_ACQUIREDEVICE	MCI_RELEASEDEVICE
MCI_CLOSE	MCI_RESUME

MCI_CUE

MCI_PAUSE

MCI_PLAY

MCI_RECORD

MCI_SEEK

MCI_SET

MCI_STOP

MCI_GROUP - Related Messages

- MCI_ACQUIREDEVICE
- MCI_CUE
- MCI_CLOSE
- MCI_PAUSE
- MCI_PLAY
- MCI_RECORD
- MCI_RELEASEDEVICE
- MCI_RESUME
- MCI_SEEK
- MCI_SET
- MCI_STOP

MCI_GROUP - Example Code

The following code illustrates how to initialize multiple devices in a group simultaneously.

```
  /***  Sample code to make a group using mciSendCommand.  ***/

MCI_GROUP_PARMS    mciGroupParameters;
ULONG              paulDeviceIDs[4];
ULONG              ulRC;
ULONG              ulGroupFlags;


  /*********************************************************************/
  /***  Assume code is here to open four devices and store their    ***/
  /***  device IDs in the array                                     ***/
  /***  paulDeviceIDs[0]...paulDeviceIDs[3]     ***/
  /*********************************************************************/


ulGroupFlags = MCI_GROUP_MAKE;               /* Make a group            */

mciGroupParameters.hwndCallback= (HWND) NULL;/* No NOTIFY will be used. */

mciGroupParameters.usGroupID   = 0;          /* This will be returned.  */

mciGroupParameters.pszGroupAlias= (PSZ) NULL;/* No alias will be used.  */

mciGroupParameters.ulNumDevices = 4;         /* Group four devices.     */

mciGroupParameters.paulDeviceID = paulDeviceIDs;/* This array contains  */
                                                   the four device IDs. */

ulRC = mciSendCommand(
        0,                           /* We don't know the group's ID yet. */
        MCI_GROUP,                   /* MCI_GROUP message.                */
        ulGroupFlags,                /* Flags for the MCI_GROUP message.  */
        (PVOID)&mciGroupParameters /* Parameters for the message.       */
        0 );                         /* User parameter.                   */
```

```
/**********************************************************************/
/***  On successful return, a group will have been created        ***/
/***  combining the four devices (whose device IDs were in the    ***/
/***  paulDeviceIDs array) into one "grouped" device.  This       ***/
/***  "grouped" device will have a device ID of its own found in  ***/
/***  the mciGroupParameters.usGroupID field.                     ***/
/**********************************************************************/
```

MCI_GROUP - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_INFO

MCI_INFO Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_INFO_PRODUCT
This flag returns a description of the particular hardware associated with a device.

CD Audio Extensions

The following additional flags apply to CD audio devices:

MCI_CD_INFO_ID
This flag returns the disc ID (8 bytes) consisting of the starting address, ending track number, and address of the lead-out track. The disc ID is generated by the CD Audio MCD and is not necessarily unique.

MCI_CD_INFO_UPC
This flag returns the disc's UPC code (serial number) if the device supports this function; otherwise it returns 0. The UPC is BCD coded. Not all discs have UPCs.

CD-XA Extensions

The following additional flags apply to CD-XA devices:

MCI_CD_INFO_UPC
This flag returns the disc's UPC code (serial number) if the device supports this function; otherwise, it returns 0. The UPC is BCD coded. Not all discs have UPCs.

MCI_INFO_FILE
This flag returns the file name of the current file.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_INFO_VIDEO_FILE
This flag returns the file name of the current digital video file used by the device.

MCI_DGV_INFO_IMAGE_FILE
This flag returns the file name of the current image file used by the device.

MCI_DGV_INFO_TEXT
This flag returns the caption of the window in which the digital video is currently displayed.

MCI_DGV_INFO_REGION
This flag returns the name of the current tuner region.

MCI_DGV_INFO_REGION_TEXT
This flag returns a description of the current tuner region.

Sequencer Extensions

The following additional flags apply to sequencer devices:

MCI_INFO_FILE
This flag returns the file name of the current file.

Videodisc Extensions

The following additional flags apply to videodisc devices:

MCI_VD_INFO_LABEL
This flag returns the videodisc label.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_INFO_FILE
This flag returns the file name of the current file.

MCI_OVLY_INFO_TEXT
This flag returns the caption of the window in which the video overlay is currently displayed.

Wave Audio Extensions

The following additional flags apply to wave audio devices:

MCI_INFO_FILE
This flag returns the file name of the current file.

MCI_INFO Parameter - pParam2

pParam2 (PMCI_INFO_PARMS)
A pointer to the MCI_INFO_PARMS data structure.

MCI_INFO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by the MMPM/2 driver for this command.

MCIERR_INVALID_CALLBACK_HANDLE
The window callback handle is not valid.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_FILE_NOT_FOUND
File not found.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
The flags cannot be used together.

MCI_INFO - Description

This message returns string information from a media device instance. This information does not describe the capabilities of the device, only static information about the device.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_INFO_PRODUCT
This flag returns a description of the particular hardware associated with a device.

CD Audio Extensions

The following additional flags apply to CD audio devices:

MCI_CD_INFO_ID
This flag returns the disc ID (8 bytes) consisting of the starting address, ending track number, and address of the lead-out track. The disc ID is generated by the CD Audio MCD and is not necessarily unique.

MCI_CD_INFO_UPC
This flag returns the disc's UPC code (serial number) if the device supports this function; otherwise it returns 0. The UPC is BCD coded. Not all discs have UPCs.

CD-XA Extensions

The following additional flags apply to CD-XA devices:

MCI_CD_INFO_UPC
This flag returns the disc's UPC code (serial number) if the device supports this function; otherwise, it returns 0. The UPC is BCD coded. Not all discs have UPCs.

MCI_INFO_FILE
This flag returns the file name of the current file.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_INFO_VIDEO_FILE
This flag returns the file name of the current digital video file used by the device.

MCI_DGV_INFO_IMAGE_FILE
This flag returns the file name of the current image file used by the device.

MCI_DGV_INFO_TEXT
This flag returns the caption of the window in which the digital video is currently displayed.

MCI_DGV_INFO_REGION
This flag returns the name of the current tuner region.

MCI_DGV_INFO_REGION_TEXT
This flag returns a description of the current tuner region.

Sequencer Extensions

The following additional flags apply to sequencer devices:

MCI_INFO_FILE
This flag returns the file name of the current file.

Videodisc Extensions

The following additional flags apply to videodisc devices:

MCI_VD_INFO_LABEL
This flag returns the videodisc label.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_INFO_FILE
This flag returns the file name of the current file.

MCI_OVLY_INFO_TEXT
This flag returns the caption of the window in which the video overlay is currently displayed.

Wave Audio Extensions

The following additional flags apply to wave audio devices:

MCI_INFO_FILE
: This flag returns the file name of the current file.

pParam2 (PMCI_INFO_PARMS)
: A pointer to the MCI_INFO_PARMS data structure.

rc (ULONG)
: Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_UNSUPPORTED_FLAG	Flag not supported by the MMPM/2 driver for this command.
MCIERR_INVALID_CALLBACK_HANDLE	The window callback handle is not valid.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_INVALID_BUFFER	Invalid return buffer given.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_MISSING_FLAG	Flag missing for this MMPM/2 command.
MCIERR_FLAGS_NOT_COMPATIBLE	The flags cannot be used together.

MCI_INFO - Remarks

The parameters and flags for this message vary according to the selected device. If the size of the buffer passed in is too small to hold all the data returned, *ulRetSize* will contain the required buffer size, the error code MCIERR_INVALID_BUFFER will be returned, and the buffer will only contain as much of the INFO data as its size permits. Only one flag can be used per MCI_INFO message; otherwise the MCIERR_FLAGS_NOT_COMPATIBLE error is returned.

MCI_INFO - Related Messages

- MCI_GETDEVCAPS

MCI_INFO - Example Code

The following code illustrates how to get the file name of the currently loaded device.

```
#define  RETBUFSIZE 128

USHORT   usDeviceID;
CHAR     InfoRet [RETBUFSIZE];            /* Return string buffer     */
MCI_INFO_PARMS  infoparms;

/* Get the file name of the currently loaded file                     */

infoparms.pszReturn = (PSZ) &InfoRet;     /* Pointer to return buffer */
infoparms.ulRetSize = RETBUFSIZE;         /* Return buffer size       */

mciSendCommand(usDeviceID,                /* Device ID                */
 MCI_INFO,                                /* MCI info message         */
 MCI_WAIT | MCI_FILE,                     /* Flags for this message   */

 (PVOID) &infoparms,                      /* Data structure           */
 0);                                      /* No user parm             */

/* NOTE: infoparms.pszReturn now contains the name
         of the current file                                          */
```

MCI_INFO - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_LOAD

MCI_LOAD Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags: The MCI_OPEN_ELEMENT and MCI_OPEN_MMIO flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_OPEN_ELEMENT

This flag specifies that an element name is included. The element name can be that of a file or a file element in a compound file. The element name is specified in the *pszElementName* field of the MCI_LOAD_PARMS data structure. If the element name does not exist or is NULL, then a temporary element is created for subsequent use. (This is the equivalent of specifying the NEW keyword with the LOAD string command.) The temporary file can be made permanent by providing a name using the MCI_SAVE message.

MCI_OPEN_MMIO

Indicates that an MMIO handle (*hmmio*) is passed in the *pszElementName* field of the open data structure. The file must have been opened through MMIO with the *ulTranslate* field of the MMIOINFO data structure set to MMIO_TRANSLATEHEADER, unless a particular MCD indicates differently.

Digital Video Extensions

MCI_READONLY

Opens the file in a read-only mode and prevents inadvertent changes to the file. When no changes to the file are allowed, the digital video driver can improve load and run-time performance, while allowing other devices to share the file for playback purposes.

This flag can only be used in conjunction with the MCI_OPEN_ELEMENT flag. Specifying the MCI_READONLY flag disables support for MCI_SAVE and MCI_RECORD.

Video Overlay Extensions

The image contained in the file is loaded into the image device element and overwrites any image currently stored there. It can be displayed using the MCI_RESTORE command.

The file is opened, accessed, and closed on this command.

If the format of the image file is not recognized as either a device specific file format or a format supported by MMIO the load fails.

Load performs an automatic *set* of the following values for:

- IMAGE BITSPERPEL
- IMAGE PELFORMAT
- IMAGE COMPRESSION
- IMAGE QUALITY
- IMAGE EXTENTS

M-Motion Overlay implementation values would be:

```
IMAGE BITSPERPEL = 21
IMAGE PELFORMAT  = yuvb
IMAGE COMPRESSION= BI_NONE
IMAGE QUALITY    = photo
IMAGE EXTENTS    = image specific
```

The previous values for these attributes are ignored.

Load also automatically *sets* IMAGE FILEFORMAT to indicate information about the original file.

Waveform Audio Extensions

MCI_READONLY

Opens the file in a read-only mode and prevents inadvertent changes to the file. When no changes to the file are allowed, the waveform audio driver can improve load and run-time performance, while allowing other devices to share the file for playback purposes.

This flag can only be used in conjunction with the MCI_OPEN_ELEMENT flag. Specifying the MCI_READONLY flag disables support for MCI_SAVE and MCI_RECORD.

MCI_LOAD Parameter - pParam2

pParam2 (PMCI_LOAD_PARMS)
A pointer to the MCI_LOAD_PARMS data structure.

MCI_LOAD Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags not compatible.
MCIERR_INSTANCE_INACTIVE	Instance inactive.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_INVALID_MEDIA_TYPE	Invalid media type given or invalid data format.
MCIERR_HARDWARE	Hardware error.
MCIERR_FILE_ATTRIBUTE	File attribute error specified.
MCIERR_UNSUPP_SAMPLESPERSEC	The hardware does not support this sampling rate
MCIERR_UNSUPP_BITSPERSAMPLE	The hardware does not support this bits per sample setting.
MCIERR_UNSUPP_CHANNELS	The hardware does not support this channel setting.
MCIERR_UNSUPP_FORMAT_MODE	The hardware does not support this format mode.
MCIERR_UNSUPP_FORMAT_TAG	The hardware does not support this format tag.

MCI_LOAD - Description

This message is used for specifying a new file or RIFF chunk to be loaded onto an already opened device instance.

ulParam1 (ULONG)
This parameter can contain any of the following flags: The MCI_OPEN_ELEMENT and MCI_OPEN_MMIO flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_OPEN_ELEMENT
This flag specifies that an element name is included. The element name can be that of a file or a file element in a compound file. The element name is specified in the *pszElementName* field of the MCI_LOAD_PARMS data structure. If the element name does not exist or is NULL, then a temporary element is created for subsequent use. (This is the equivalent of specifying the NEW keyword with the LOAD string command.) The temporary file can be made permanent by providing a name using the MCI_SAVE message.

MCI_OPEN_MMIO
Indicates that an MMIO handle (*hmmio*) is passed in the *pszElementName* field of the open data structure. The file must have been opened through MMIO with the *ulTranslate* field of the MMIOINFO data structure set to MMIO_TRANSLATEHEADER, unless a particular MCD indicates differently.

Digital Video Extensions

MCI_READONLY
Opens the file in a read-only mode and prevents inadvertent changes to the file. When no changes to the file are allowed, the digital video driver can improve load and run-time performance, while allowing other devices to share the file for playback purposes.

This flag can only be used in conjunction with the MCI_OPEN_ELEMENT flag. Specifying the MCI_READONLY flag disables support for MCI_SAVE and MCI_RECORD.

Video Overlay Extensions

The image contained in the file is loaded into the image device element and overwrites any image currently stored there. It can be displayed using the MCI_RESTORE command.

The file is opened, accessed, and closed on this command.

If the format of the image file is not recognized as either a device specific file format or a format supported by MMIO the load fails.

Load performs an automatic *set* of the following values for:

- IMAGE BITSPERPEL
- IMAGE PELFORMAT
- IMAGE COMPRESSION
- IMAGE QUALITY
- IMAGE EXTENTS

M-Motion Overlay implementation values would be:

```
IMAGE BITSPERPEL = 21
IMAGE PELFORMAT  = yuvb
IMAGE COMPRESSION= BI_NONE
IMAGE QUALITY    = photo
IMAGE EXTENTS    = image specific
```

The previous values for these attributes are ignored.

Load also automatically *sets* IMAGE FILEFORMAT to indicate information about the original file.

Waveform Audio Extensions

MCI_READONLY
Opens the file in a read-only mode and prevents inadvertent changes to the file. When no changes to the file are allowed, the waveform audio driver can improve load and run-time performance, while allowing other devices to

share the file for playback purposes.

This flag can only be used in conjunction with the MCI_OPEN_ELEMENT flag. Specifying the MCI_READONLY flag disables support for MCI_SAVE and MCI_RECORD.

pParam2 (PMCI_LOAD_PARMS)
A pointer to the MCI_LOAD_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags not compatible.
MCIERR_INSTANCE_INACTIVE	Instance inactive.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_INVALID_MEDIA_TYPE	Invalid media type given or invalid data format.
MCIERR_HARDWARE	Hardware error.
MCIERR_FILE_ATTRIBUTE	File attribute error specified.
MCIERR_UNSUPP_SAMPLESPERSEC	The hardware does not support this sampling rate
MCIERR_UNSUPP_BITSPERSAMPLE	The hardware does not support this bits per sample setting.
MCIERR_UNSUPP_CHANNELS	The hardware does not support this channel setting.
MCIERR_UNSUPP_FORMAT_MODE	The hardware does not support this format mode.
MCIERR_UNSUPP_FORMAT_TAG	The hardware does not support this format tag.

MCI_LOAD - Remarks

When an existing media element is loaded into a device, the settings for the device will change if they are overridden by the settings required by the media element.

If a new media element is created by loading a nonexistent media element, the new media element should be created with default settings for the particular device.

MCI_LOAD - Default Processing

MCI_OPEN_ELEMENT is the default for the MCI_LOAD message.

MCI_LOAD - Related Messages

- MCI_OPEN

MCI_LOAD - Example Code

The following code illustrates how to load an existing file into the waveaudio device.

```
USHORT         usDeviceID;
MCI_LOAD_PARMS mlp;

mlp.hwndCallback = (HWND) NULL;   /* Not required if waiting */
strcpy(mlp.pszElementName, "oinker.wav");
                                  /* File name to load        */

mciSendCommand( usDeviceID,       /* Device ID                */
 MCI_LOAD,                        /* MCI load message         */
 MCI_WAIT | MCI_OPEN_ELEMENT,     /* Flags for this message   */
 (PVOID) &mlp,                    /* Data structure           */
 0);                              /* No user parm             */
```

MCI_LOAD - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Related Messages
Example Code
Glossary

MCI_MASTERAUDIO

MCI_MASTERAUDIO Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: The MCI_NOTIFY flag is not valid for this message.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERYCURRENTSETTING
This flag queries the current setting of the indicated audio attribute.

MCI_QUERYSAVEDSETTING
This flag queries the saved setting of the indicated audio attribute.

MCI_SAVESETTING
This flag saves the current setting of the indicated audio attribute to the INI file.

MCI_MASTERVOL
This flag sets the system master volume level as a percentage. If a number greater than 100 is given then 100 will be used as the master volume setting and no error will be returned.

MCI_SPEAKERS
This flag sets the output to speakers.

MCI_HEADPHONES
This flag sets the output to headphones.

MCI_ON
This flag sets the output on or enabled. This flag must be used in conjunction with the MCI_SPEAKERS or MCI_HEADPHONES flag.

MCI_OFF
This flag sets output off or disabled. This flag must be used in conjunction with the MCI_SPEAKERS or MCI_HEADPHONES flag.

MCI_MASTERAUDIO Parameter - pParam2

pParam2 (PMCI_MASTERAUDIO_PARMS)
A pointer to the MCI_MASTERAUDIO_PARMS data structure.

MCI_MASTERAUDIO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_MASTERAUDIO - Description

This message provides support for setting and retrieving system-wide audio control settings.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

Note: The MCI_NOTIFY flag is not valid for this message.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_QUERYCURRENTSETTING
This flag queries the current setting of the indicated audio attribute.

MCI_QUERYSAVEDSETTING
This flag queries the saved setting of the indicated audio attribute.

MCI_SAVESETTING
This flag saves the current setting of the indicated audio attribute to the INI file.

MCI_MASTERVOL
This flag sets the system master volume level as a percentage. If a number greater than 100 is given then 100 will be used as the master volume setting and no error will be returned.

MCI_SPEAKERS
This flag sets the output to speakers.

MCI_HEADPHONES
This flag sets the output to headphones.

MCI_ON
This flag sets the output on or enabled. This flag must be used in conjunction with the MCI_SPEAKERS or MCI_HEADPHONES flag.

MCI_OFF
This flag sets output off or disabled. This flag must be used in conjunction with the MCI_SPEAKERS or MCI_HEADPHONES flag.

pParam2 (PMCI_MASTERAUDIO_PARMS)
A pointer to the MCI_MASTERAUDIO_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_INVALID_FLAG

Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_MASTERAUDIO - Remarks

Two levels of volume control are provided: system wide and device-instance specific. Where as the MCI_SET command affects only one specific device opened by an application, the MCI_MASTERAUDIO command affects all open logical devices in the system.

When opened, each logical device queries these values and automatically adjusts its settings accordingly. Only applications that are intended to replace the Volume Control application should reference and modify these settings.

MCI_MASTERAUDIO - Example Code

The following code illustrates how to get the current master volume setting.

```
ULONG mastervolume;              /* Set to master volume
                                    percentage by this example      */
BOOL  speakers_on;               /* Set to TRUE if speaker
                                    output is enabled               */
USHORT usDeviceID;
MCI_MASTERAUDIO_PARMS masteraudioparms;

                                 /* Get current system master
                                    volume setting                  */

mciSendCommand(usDeviceID,       /* Device                          */
 MCI_MASTERAUDIO,                /* Master audio message            */
 MCI_WAIT | MCI_QUERYCURRENTSETTING | MCI_MASTERVOL,
                                 /* Flags for this message          */
 (PVOID) &masteraudioparms,      /* Data structure                  */
 0);                             /* User parm                       */

mastervolume = masteraudioparms.ulReturn;


                                 /* Get current system speaker
                                    enable status                   */

mciSendCommand(usDeviceID,       /* Device                          */
 MCI_MASTERAUDIO,                /* Master audio message            */
 MCI_WAIT | MCI_QUERYCURRENTSETTING | MCI_SPEAKERS,
 (PVOID) &masteraudioparms,       /* Flags for this message         */
 0);                              /* Data structure user parm       */
speakers_on = masteraudioparms.ulReturn;
```

MCI_MASTERAUDIO - Topics

Select an item:
Description
Returns

Remarks
Example Code
Glossary

MCI_MIXNOTIFY

MCI_MIXNOTIFY Parameter - ulParam1

ulParam1 (ULONG)
The following flags can be used with an amplifier-mixer device.

MCI_NOTIFY
A notification message is posted to the window specified in the *hwndCallback* parameter of the data structure identified by *pParam2* when the action indicated by this message is completed.

MCI_WAIT
Control is not returned until the action indicated by this message is completed.

MCI_MIXNOTIFY_ON
Turns mix notifications on. A valid window handle must be specified in the *hwndCallback* field of MCI_GENERIC_PARMS. If an invalid handle is specified, MCIERR_INVALID_CALLBACK_HANDLE will be returned.

MCI_MIXNOTIFY_OFF
Turns mix notifications off.

MCI_MIXNOTIFY Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the MCI_GENERIC_PARMS data structure.

MCI_MIXNOTIFY Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
Command completed successfully.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag is not supported by this device.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCI_MIXNOTIFY - Description

This message notifies an application of every mixer attribute change if the application registers for the event. When a mixer attribute is changed, an MM_MCIEVENT message is sent to the requesting application.

ulParam1 (ULONG)
The following flags can be used with an amplifier-mixer device.

MCI_NOTIFY
A notification message is posted to the window specified in the *hwndCallback* parameter of the data structure identified by *pParam2* when the action indicated by this message is completed.

MCI_WAIT
Control is not returned until the action indicated by this message is completed.

MCI_MIXNOTIFY_ON
Turns mix notifications on. A valid window handle must be specified in the *hwndCallback* field of MCI_GENERIC_PARMS. If an invalid handle is specified, MCIERR_INVALID_CALLBACK_HANDLE will be returned.

MCI_MIXNOTIFY_OFF
Turns mix notifications off.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the MCI_GENERIC_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
Command completed successfully.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag is not supported by this device.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCI_MIXNOTIFY - Remarks

When the MM_MCIEVENT message is received, the *usEventCode* field of *MsgParam1* contains MCI_MIXEVENT. The *pEventData* field of *MsgParam2* contains a pointer to MCI_MIXEVENT_PARMS. MCI_MIXEVENT_PARMS allows applications to determine the device type that caused the change, the attribute that caused the change (volume, bass, treble, and so on), and the new value of the attribute. A mixer event will also be sent when a connector has been enabled or disabled. The *ulConnectorType* and *ulConnectorIndex* fields will indicate the connector that changed and *ulConnStatus* contains either MCI_TRUE if the connector is enabled or MCI_FALSE if the connector is disabled. If a connector has been modified, *ulFlags* will contain MCI_MIX_CONNECTOR. If an attribute has been changed, *ulFlags* will contain MCI_MIX_ATTRIBUTE.

Note: An application must *not* set an audio attribute while processing the MM_MCIEVENT message. Otherwise a terminal loop will result.

MCI_MIXNOTIFY - Example Code

The following example illustrates how an application can set up notification for every audio attribute change.

```
MCI_GENERIC_PARMS  mixevent;

mixevent.hwndCallback = hwndMixer;

if (hMixer)
  {
  mciSendCommand(hMixer,
  MCI_MIXNOTIFY,                   /* MCI mixer message */
  MCI_WAIT | MCI_MIXNOTIFY_ON,     /* Flags for this message */
  (PVOID)&mixevent,                /* Data structure */
  0);                              /* No user parm */
```

MCI_MIXNOTIFY - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_MIXSETUP

MCI_MIXSETUP Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message is posted to the window specified in the *hwndCallback* field of the data structure identified by *pParam2* when the action indicated by this message is completed.

MCI_WAIT
Control is not returned until the action indicated by this message is completed.

MCI_MIXSETUP_INIT
Initializes the mixer for the correct mode according to the value specified in the *ulFormatMode* field of MCI_MIXSETUP_PARMS.

MCI_MIXSETUP_DEINIT
Deinitializes the mixer.

MCI_MIXSETUP_QUERYMODE
Queries a device to see if a specific mode is supported.

MCI_MIXSETUP Parameter - pParam2

pParam2 (PMCI_MIXSETUP_PARMS)
A pointer to the MCI_MIXSETUP_PARMS data structure.

MCI_MIXSETUP Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
Command completed successfully.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_MODE
Device mode invalid for this command.

MCIERR_INVALID_DEVICE_TYPE
Mixer does not support the requested device type.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCI_MIXSETUP - Description

This message informs the mixer device that the application wishes to read or write buffers directly and sets up the device in the correct mode (for example, PCM, MPEG audio or MIDI).

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message is posted to the window specified in the *hwndCallback* field of the data structure identified by *pParam2* when the action indicated by this message is completed.
MCI_WAIT	Control is not returned until the action indicated by this message is completed.
MCI_MIXSETUP_INIT	Initializes the mixer for the correct mode according to the value specified in the *ulFormatMode* field of MCI_MIXSETUP_PARMS.
MCI_MIXSETUP_DEINIT	Deinitializes the mixer.
MCI_MIXSETUP_QUERYMODE	Queries a device to see if a specific mode is supported.

pParam2 (PMCI_MIXSETUP_PARMS)
A pointer to the MCI_MIXSETUP_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	Command completed successfully.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_INVALID_MODE	Device mode invalid for this command.
MCIERR_INVALID_DEVICE_TYPE	Mixer does not support the requested device type.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.

MCI_MIXSETUP - Remarks

On input, the application must fill in the *ulDeviceType* field of the MCI_MIXSETUP_PARMS structure to inform the mixer of the media type it will be sending. The application must also fill in the *pmixEvent* field of the MCI_MIXSETUP_PARMS structure with a callback function for the mixer to call when it is finished writing or reading a buffer.

If the call is successful, the mixer will update the *pmixWrite* and *pmixRead* fields so that the application can write or read buffers to or from the mixer. In addition, the mixer will update the *ulBufferSize* and *ulNumBuffers* fields with the *suggested* buffer size and number of buffers to use with the requested setup. The application does not have to use these suggested values as they are simply recommendations.

If the mixer has already been initialized with MCI_MIXSETUP and MCI_MIXSETUP is called again, MCIERR_INVALID_MODE will be returned.

After MCI_MIXSETUP has been successfully called, you can use MCI_BUFFER to allocate or deallocate memory for communcation with the audio device.

MCI_MIXSETUP - Related Messages

- MCI_BUFFER

MCI_MIXSETUP - Example Code

The following code illustrates using MCI_MIXSETUP to prepare the audio device for 16-bit, 22050 kHz, stereo mode.

```
    memset( &MixSetupParms, '\0', sizeof( MCI_MIXSETUP_PARMS ) );

  MixSetupParms.ulBitsPerSample = 16;
  MixSetupParms.ulFormatTag = MCI_WAVE_FORMAT_PCM;
  MixSetupParms.ulSamplesPerSec = 22050;
  MixSetupParms.ulChannels = 2;  /* Stereo */
  MixSetupParms.ulBitsPerSample = 16;
  MixSetupParms.ulFormatMode = MCI_PLAY;
  MixSetupParms.ulDeviceType = MCI_DEVTYPE_WAVEFORM_AUDIO;

   /* The mixer will inform us of entry points to */
   /* read/write buffers to and also give us a    */
   /* handle to use with these entry points.      */

  MixSetupParms.pmixEvent = MyEvent;

    rc = mciSendCommand( usDeviceID,
                     MCI_MIXSETUP,
                     MCI_WAIT | MCI_MIXSETUP_INIT,
                     ( PVOID ) &MixSetupParms,
                     0 );
```

MCI_MIXSETUP - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_OPEN

MCI_OPEN Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags. MCI_OPEN_ELEMENT and MCI_OPEN_MMIO are mutually exclusive flags.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_DOS_QUEUE
This flag specifies that window handles passed in for this device instance will be treated as OS/2 Control Program queue handles.

MCI_OPEN_ALIAS
This flag specifies that the *pszAlias* field of the open structure contains an alias for this device instance. This alias can then be used on subsequent commands using the string interface.

MCI_OPEN_ELEMENT
This flag specifies that an element name is included. The element name can be that of a file or a file element in a compound file. The element name is specified in the *pszElementName* field of the open data structure. If the element name does not exist or is NULL, then a temporary element is created for subsequent use. The temporary file can be made permanent by providing a name using the MCI_SAVE message.

MCI_OPEN_MMIO
This flag specifies that an MMIO handle (*hmmio*) is passed in the *pszElementName* field of the open data structure. The file must have been opened through MMIO with *ulTranslate* of the MMIOINFO data structure set to MMIO_TRANSLATEHEADER, unless a particular MCD indicates differently.

MCI_OPEN_PLAYLIST
This flag indicates that the *pszElementName* field of the open data structure contains a pointer to a memory playlist structure.

MCI_OPEN_READONLY
This flag specifies that the file is to be opened in read-only mode. The load and run-time performance for the wave audio and digital video devices can be improved by specifying this flag. This flag can only be used in conjunction with the MCI_OPEN_ELEMENT or MCI_OPEN_MMIO flags. By specifying this flag, MCI_RECORD and MCI_SAVE are automatically disabled.

MCI_OPEN_SHAREABLE
This flag specifies that the device instance is to be opened in a fully shareable mode. Omitting this flag causes the device instance to be opened for exclusive use.

MCI_OPEN_TYPE_ID
This flag specifies that the *pszDeviceType* field of the open data structure is to be interpreted as follows. The low-order word is a standard device type and the high-order word is the ordinal index for the device. If MCI_OPEN_TYPE_ID is specified and the index is 0, the default device will be opened. If MCI_OPEN_TYPE_ID is not specified and the *pszDeviceType* field is not NULL, the media control interface will attempt to open the device specified by *pszDeviceType*. If MCI_OPEN_TYPE_ID is not specified, *pszDeviceType* is NULL, and the MCI_OPEN_ELEMENT flag is specified, the system attempts to select and open a device based on the element extension or EA type of the file specified in the *pszElementName* field of the open data structure.

Digital Video Extensions

The following flags apply to digital video devices:

MCI_DGV_OPEN_PARENT
This flag indicates that the *hwndParent* field of the open data structure contains a valid parent window handle. If this flag is not specified, HWND_DESKTOP is assumed to be the parent window handle.

Video Overlay Extensions

The following flag applies to video overlay devices:

MCI_OVLY_OPEN_PARENT
This flag indicates that the *hwndParent* field of the open data structure contains a valid parent window handle. If this flag is not specified, HWND_DESKTOP is assumed to be the parent window handle.

MCI_OPEN Parameter - pParam2

pParam2 (PMCI_OPEN_PARMS)
A pointer to the MCI_OPEN_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_OPEN_PARMS	A pointer to the MCI_AMP_OPEN_PARMS data structure.
PMCI_DGV_OPEN_PARMS	A pointer to the MCI_DGV_OPEN_PARMS data structure.
PMCI_OVLY_OPEN_PARMS	A pointer to the MCI_OVLY_OPEN_PARMS data structure.

MCI_OPEN Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_UNSUPPORTED_FLAG	Flag not supported by this MMPM/2 driver for this command.
MCIERR_DEVICE_LOCKED	Device is locked.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_INI_FILE	MMPM2.INI file error.

MCIERR_OVLY_MAX_OPEN_LIMIT
: Opened maximum limit.

MCIERR_INVALID_MEDIA_TYPE
: Invalid media type given.

MCIERR_HARDWARE
: Hardware error.

MCIERR_FILE_ATTRIBUTE
: File attribute error specified.

MCIERR_NO_DEVICEDRIVER
: There was no device driver found or it is not operational.

MCIERR_UNSUPP_SAMPLESPERSEC
: The hardware does not support this sampling rate

MCIERR_UNSUPP_BITSPERSAMPLE
: The hardware does not support this bits per sample setting.

MCIERR_UNSUPP_CHANNELS
: The hardware does not support this channel setting.

MCIERR_UNSUPP_FORMAT_MODE
: The hardware does not support this format mode.

MCIERR_UNSUPP_FORMAT_TAG
: The hardware does not support this format tag.

MMIOERR_ACCESS_DENIED
The file cannot be opened.

MCI_OPEN - Description

This message is used to open or create a new device instance.

ulParam1 (ULONG)
This parameter can contain any of the following flags. MCI_OPEN_ELEMENT and MCI_OPEN_MMIO are mutually exclusive flags.

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_DOS_QUEUE
: This flag specifies that window handles passed in for this device instance will be treated as OS/2 Control Program queue handles.

MCI_OPEN_ALIAS
: This flag specifies that the *pszAlias* field of the open structure contains an alias for this device instance. This alias can then be used on subsequent commands using the string interface.

MCI_OPEN_ELEMENT
: This flag specifies that an element name is included. The element name can be that of a file or a file element in a compound file. The element name is specified in the *pszElementName* field of the open data structure. If the element name does not exist or is NULL, then a temporary element is created for subsequent use. The temporary file can be made permanent by providing a name using the MCI_SAVE message.

MCI_OPEN_MMIO
: This flag specifies that an MMIO handle (*hmmio*) is passed in the *pszElementName* field of the open data structure. The file must have been opened through MMIO with *ulTranslate* of the MMIOINFO data structure set to MMIO_TRANSLATEHEADER, unless a particular MCD indicates differently.

MCI_OPEN_PLAYLIST
: This flag indicates that the *pszElementName* field of the open data structure contains a pointer to a memory playlist structure.

MCI_OPEN_READONLY
: This flag specifies that the file is to be opened in read-only mode. The load and run-time performance for the wave audio and digital video devices can be improved by specifying this flag. This flag can only be used in conjunction with the MCI_OPEN_ELEMENT or MCI_OPEN_MMIO flags. By specifying this flag, MCI_RECORD and MCI_SAVE are automatically disabled.

MCI_OPEN_SHAREABLE
: This flag specifies that the device instance is to be opened in a fully shareable mode. Omitting this flag causes the device instance to be opened for exclusive use.

MCI_OPEN_TYPE_ID
: This flag specifies that the *pszDeviceType* field of the open data structure is to be interpreted as follows. The low-order word is a standard device type and the high-order word is the ordinal index for the device. If MCI_OPEN_TYPE_ID is specified and the index is 0, the default device will be opened. If MCI_OPEN_TYPE_ID is not specified and the *pszDeviceType* field is not NULL, the media control interface will attempt to open the device specified by *pszDeviceType*. If MCI_OPEN_TYPE_ID is not specified, *pszDeviceType* is NULL, and the MCI_OPEN_ELEMENT flag is specified, the system attempts to select and open a device based on the element extension or EA type of the file specified in the *pszElementName* field of the open data structure.

Digital Video Extensions

The following flags apply to digital video devices:

MCI_DGV_OPEN_PARENT
: This flag indicates that the *hwndParent* field of the open data structure contains a valid parent window handle. If this flag is not specified, HWND_DESKTOP is assumed to be the parent window handle.

Video Overlay Extensions

The following flag applies to video overlay devices:

MCI_OVLY_OPEN_PARENT
: This flag indicates that the *hwndParent* field of the open data structure contains a valid parent window handle. If this flag is not specified, HWND_DESKTOP is assumed to be the parent window handle.

pParam2 (PMCI_OPEN_PARMS)
: A pointer to the MCI_OPEN_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_OPEN_PARMS
: A pointer to the MCI_AMP_OPEN_PARMS data structure.

PMCI_DGV_OPEN_PARMS
: A pointer to the MCI_DGV_OPEN_PARMS data structure.

PMCI_OVLY_OPEN_PARMS
: A pointer to the MCI_OVLY_OPEN_PARMS data structure.

rc (ULONG)
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER

Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_UNSUPPORTED_FLAG	Flag not supported by this MMPM/2 driver for this command.
MCIERR_DEVICE_LOCKED	Device is locked.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_INI_FILE	MMPM2.INI file error.
MCIERR_OVLY_MAX_OPEN_LIMIT	Opened maximum limit.
MCIERR_INVALID_MEDIA_TYPE	Invalid media type given.
MCIERR_HARDWARE	Hardware error.
MCIERR_FILE_ATTRIBUTE	File attribute error specified.
MCIERR_NO_DEVICEDRIVER	There was no device driver found or it is not operational.
MCIERR_UNSUPP_SAMPLESPERSEC	The hardware does not support this sampling rate
MCIERR_UNSUPP_BITSPERSAMPLE	The hardware does not support this bits per sample setting.
MCIERR_UNSUPP_CHANNELS	The hardware does not support this channel setting.
MCIERR_UNSUPP_FORMAT_MODE	The hardware does not support this format mode.
MCIERR_UNSUPP_FORMAT_TAG	The hardware does not support this format tag.
MMIOERR_ACCESS_DENIED	The file cannot be opened.

MCI_OPEN - Remarks

Case is ignored in the device name, but there must not be any leading or trailing blanks. Note that the device type is the *pszDeviceType* element of the open data structure, but it does not have a corresponding flag because it is required and does not have a command-string parameter. Also, if automatic type selection is desired (through the extensions or EA section or INI), the file name (including the file extension) must be passed in the *pszElementName* field, the *pszDeviceType* field must be left NULL, and the MCI_OPEN_ELEMENT flag must be set.

If a parent window handle is specified, but the window handle is invalid, the overlay device opens successfully, but uses HWND_DESKTOP as its parent.

MCI_OPEN - Default Processing

If the MCI_OPEN_SHAREABLE flag is not specified, the device instance is opened for exclusive use.

If the MCI_OPEN_TYPE_ID flag is not specified and the *pszDeviceType* field of the open data structure is not NULL, the media control interface attempts to open the device specified by the *pszDeviceType* string. If *pszDeviceType* is NULL and MCI_OPEN_ELEMENT flag is specified, the media control interface attempts to select and open a device based on the element extension or EA type of the file specified in the *pszElementName* field of the open data structure.

If *pszDeviceType* is a device type ID with a NULL ordinal or a string device name with no ordinals, then the default device of the specified type is opened. The default device can be selected using the Multimedia Setup application.

MCI_OPEN - Related Messages

- MCI_LOAD

MCI_OPEN - Example Code

The following code illustrates how to open a waveaudio device instance by specifying SPEECH.WAV.

```
/* Open a waveaudio device context, specifying the element
   "speech.wav".

ULONG          rc;
MCI_OPEN_PARMS mop;

mop.hwndCallback = (HWND) NULL;      /* N/A - we're waiting       */
mop.usDeviceID = (USHORT) NULL;      /* This is returned          */
mop.pszDeviceType = (PSZ) NULL;      /* using default device type */
mop.pszElementName = (PSZ) "speech.wav"
                                     /* File name to open         */
rc = mciSendCommand( 0,
    MCI_OPEN,                        /* MCI open message          */
    MCI_WAIT | MCI_OPEN_ELEMENT |
    MCI_OPEN_SHAREABLE,              /* Flags for this message    */
    (ULONG) &mop,                    /* Data structure            */
    0);                              /* No user parm              */
if (LOUSHORT(rc) == MCIERR_SUCCESS)
  {
   usDeviceID = mop.usDeviceID;      /* Return device ID          */
  }
```

MCI_OPEN - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Related Messages

Example Code
Glossary

MCI_PASTE

MCI_PASTE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
Marks the beginning position of the paste operation. This position is specified in the *ulFrom* field of the MCI_EDIT_PARMS data structure. If MCI_FROM is not specified, the paste operation begins from the current position.

MCI_TO
Marks the ending position of the paste. The pasted data *replaces* data from the FROM position (or the current position if MCI_FROM is not specified) to the TO position.

If MCI_TO is not specified, the end of file is assumed and the pasted data is *inserted* beginning at the FROM position (or the current position if MCI_FROM is not specified).

MCI_CONVERT_FORMAT
Converts data in the clipboard to a destination format.

MCI_TO_BUFFER
Places data from the clipboard into the application's buffer. If this flag is not specified, the information is placed in a file.

MCI_FROM_BUFFER
Places data from the application's buffer into the file. If this flag is not specified, the clipboard is used as the source.

MCI_PASTE Parameter - pParam2

pParam2 (PMCI_EDIT_PARMS)
A pointer to the MCI_EDIT_PARMS structure.

MCI_PASTE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	The paste was successful.
MCIERR_INVALID_BUFFER	The buffer is too small to hold data.
MCIERR_CANNOT_WRITE	The file was not opened with write access.
MCIERR_OUTOFRANGE	The units are out of the range.
MCIERR_INVALID_MEDIA	The clipboard format is not valid.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_OUT_OF_MEMORY	There is insufficient memory to perform the operation requested.
MCIERR_CLIPBOARD_EMPTY	No recognizable information is in the clipboard.
MCIERR_CANNOT_CONVERT	Unable to convert clipboard information to destination.
MMIOERR_CLIPBRD_EMPTY	There is no compatible data in the clipboard for use by the paste operation.
MMIOERR_CLIPBRD_ERROR	An unrecoverable error occurred while attempting to access the clipboard.
MMIOERR_INCOMPATIBLE_DATA	The data in the clipboard cannot be pasted into this file because the characteristics of either the video or audio data, or both, do not match the characteristics of the target file.

MCI_PASTE - Description

This message pastes data from the clipboard or application buffer into a file starting at the from position. Following a paste operation the media position is at the end of the pasted data. However, after pasting into a *new* file, the media position will be at 0.

The MCI_CONVERT_FORMAT flag is not supported by the digital video device.

ulParam1 (ULONG)

This parameter can contain any of the following flags:

MCI_NOTIFY

A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT

Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM

Marks the beginning position of the paste operation. This position is specified in the *ulFrom* field of the MCI_EDIT_PARMS data structure. If MCI_FROM is not specified, the paste operation begins from the current position.

MCI_TO

Marks the ending position of the paste. The pasted data *replaces* data from the FROM position (or the current position if MCI_FROM is not specified) to the TO position.

If MCI_TO is not specified, the end of file is assumed and the pasted data is *inserted* beginning at the FROM position (or the current position if MCI_FROM is not specified).

MCI_CONVERT_FORMAT

Converts data in the clipboard to a destination format.

MCI_TO_BUFFER

Places data from the clipboard into the application's buffer. If this flag is not specified, the information is placed in a file.

MCI_FROM_BUFFER

Places data from the application's buffer into the file. If this flag is not specified, the clipboard is used as the source.

pParam2 (PMCI_EDIT_PARMS)

A pointer to the MCI_EDIT_PARMS structure.

rc (ULONG)

Return codes indicating success or type of failure:

MCIERR_SUCCESS

The paste was successful.

MCIERR_INVALID_BUFFER

The buffer is too small to hold data.

MCIERR_CANNOT_WRITE

The file was not opened with write access.

MCIERR_OUTOFRANGE

The units are out of the range.

MCIERR_INVALID_MEDIA

The clipboard format is not valid.

MCIERR_INVALID_DEVICE_ID

The device ID is not valid.

MCIERR_MISSING_PARAMETER

Required parameter is missing.

MCIERR_INVALID_FLAG

Flag (*ulParam1*) is invalid.

MCIERR_UNSUPPORTED_FLAG

Given flag is unsupported for this device.

MCIERR_INSTANCE_INACTIVE

The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to perform the operation requested.

MCIERR_CLIPBOARD_EMPTY
No recognizable information is in the clipboard.

MCIERR_CANNOT_CONVERT
Unable to convert clipboard information to destination.

MMIOERR_CLIPBRD_EMPTY
There is no compatible data in the clipboard for use by the paste operation.

MMIOERR_CLIPBRD_ERROR
An unrecoverable error occurred while attempting to access the clipboard.

MMIOERR_INCOMPATIBLE_DATA
The data in the clipboard cannot be pasted into this file because the characteristics of either the video or audio data, or both, do not match the characteristics of the target file.

MCI_PASTE - Remarks

The units of the MCI_FROM and MCI_TO parameters must be supplied in the selected time format. If neither MCI_FROM or MCI_TO are specified, MCI_PASTE inserts the clipboard contents at the current position.

The MCI_CONVERT_FORMAT converts what was in the clipboard to the destination file format. The following conversions can be done:

Settings	**Conversions**
Channels	Mono to stereo, stereo to mono.
Sampling rate	11025, 22050, or 44100 to 11025, 22050, or 44100.
Data Type	16-bit to 8-bit, 8-bit to 16-bit.

Note: No smoothing is performed on the paste.

If a paste interrupts an in-progress operation, such as play, the command is aborted and an MM_MCINOTIFY message is sent to the application.

The implementation of the paste operation for AVI movie files does not support data transformations. The AVI movie file being pasted into must have the same video and audio characteristics as the file from which the clipboard data was obtained. (The video data must have the same nominal frame rate, frame size, and use the same decompressor; the audio data must be the same type and must match in number of channels, samples per second, and bits per sample.) MMIOERR_INCOMPATIBLE_DATA is returned if the clipboard data does not match the data in the target file.

Edited Audio/Video Interleaved (AVI) movie files cannot always be saved with their original name after the paste operation. If the clipboard contains a reference to data that would be erased during saving or if another instance of the digital video device has a pending paste operation which depends on this data, the file cannot be saved unless a new file name has been provided. If a new file name is not provided, MMIOERR_NEED_NEW_FILENAME is returned by the AVI I/O procedure and a temporary file is created to save the edited movie.

Note: AVI is the only video file format supporting editing commands.

Waveaudio Specific

If MCI_FROM_BUFFER or MCI_TO_BUFFER are used, the *pHeader* field of MCI_EDIT_PARMS must contain a pointer to an MMAUDIOHEADER structure. The *ulBufLen* field of MCI_EDIT_PARMS must be filled in.

MCI_PASTE - Related Messages

- MCI_COPY
- MCI_CUT
- MCI_DELETE
- MCI_UNDO
- MCI_REDO

MCI_PASTE - Example Code

The following code illustrates pasting data from the clipboard into the current file position.

```
USHORT                usDeviceID;
MCI_EDIT_PARMS        mep;

mep.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,
                MCI_PASTE,
                MCI_NOTIFY,
                &mep,
                0 );
```

MCI_PASTE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_PAUSE

MCI_PAUSE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT

Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_PAUSE Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_PAUSE Return Value - rc

rc (ULONG)
Return codes indicating success or failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_PAUSE - Description

This message is sent to suspend playback or recording.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
: The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
: The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
: Unsupported function.

MCIERR_INVALID_FLAG
: Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCI_PAUSE - Remarks

The MCI_RESUME message is used to return the device to the previous playback or recording operation from the paused state to the parameters of the previous operation that remain in effect.

If the device is paused and MCI_PLAY or MCI_RECORD is issued, the previous action is superseded and from and to parameters are used from the newly issued message.

MCI_PAUSE - Related Messages

- MCI_PLAY
- MCI_RECORD
- MCI_RESUME

MCI_PAUSE - Example Code

The following code illustrates how to pause a device and request notification when the operation is completed.

```
/* Pause the device, requesting notification when operation completes */

#define UP_PAUSE 1
```

```
USHORT usDeviceID;
HWND hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms;        /* Generic message
                                             parms structure   */

                 /* Assign hwndCallback the handle to the PM Window */

mciGenericParms.hwndCallback = hwndMyWindow;

mciSendCommand(usDeviceID,      /* Device ID                        */
               MCI_PAUSE,       /* MCI pause message                */
               MCI_NOTIFY,      /* Flag for this message            */
               (PVOID) &mciGenericParms,     /* Data structure      */
               UP_PAUSE);       /* User parameter to be returned
                                   on notification message          */
```

MCI_PAUSE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_PLAY

MCI_PLAY Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
This flag indicates that the *ulFrom* field of the play data structure is to be used as the starting position for the play operation. If this flag is not set, the current position is assumed.

MCI_TO
This flag indicates that the *ulTo* field of the play data structure is to be used as the ending position of the play operation. If this flag is not set, playback continues to the end of the media or segment, as defined by the device. If the to position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_PLAY_SPEED
: This flag adds a speed parameter. The units are specified by the currently set speed format. The speed value is in the *ulSpeed* field in the play data structure.

MCI_DGV_PLAY_REVERSE
: This flag specifies to play in reverse.

MCI_DGV_PLAY_FAST
: This flag specifies to play at the fast rate (twice the normal recorded playback rate).

MCI_DGV_PLAY_SCAN
: Specifies to scan. Scan usually means to play as quickly as possible, with audio disabled.

MCI_DGV_PLAY_SLOW
: This flag specifies to play at the slow rate (half the normal recorded playback rate).

MCI_DGV_PLAY_REPEAT
: This flag specifies that the play operation be repeated until the command is superseded by another command or aborted.

: This flag is not supported by the digital video MCD.

Videodisc Extensions

The following additional flags apply to videodisc devices. MCI_VD_PLAY_REVERSE and MCI_VD_PLAY_SCAN are mutually exclusive. Only one of the other flags is allowed with this message.

MCI_VD_PLAY_REVERSE
: This flag specifies to play in reverse.

MCI_VD_PLAY_FAST
: This flag specifies to play at the fast rate.

MCI_VD_PLAY_SCAN
: This flag specifies to scan. Scan usually means to play as fast as possible, with audio disabled.

MCI_VD_PLAY_SPEED
: This flag adds a speed parameter. The units are specified by the currently set speed format. The speed value is in the *ulSpeed* field of the play data structure.

MCI_VD_PLAY_SLOW
: This flag specifies to play at the slow rate.

MCI_PLAY Parameter - pParam2

pParam2 (PMCI_PLAY_PARMS)
A pointer to an MCI_PLAY_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_PLAY_PARMS
: A pointer to an MCI_DGV_PLAY_PARMS data structure.

PMCI_VD_PLAY_PARMS
: A pointer to an MCI_VD_PLAY_PARMS data structure.

MCI_PLAY Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_MEDIA_CHANGED	The required media has changed.
MCIERR_DEVICE_NOT_READY	The device is not ready.
MCIERR_INVALID_DEVICE_ID	The device id is not VALID.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_UNSUPPORTED_FUNCTION	Unsupported function.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_OUTOFRANGE	Units are out of range.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_CHANNEL_OFF	Primary channel is off.

MCI_PLAY - Description

This message is sent to begin playback.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
: This flag indicates that the *ulFrom* field of the play data structure is to be used as the starting position for the play operation. If this flag is not set, the current position is assumed.

MCI_TO
: This flag indicates that the *ulTo* field of the play data structure is to be used as the ending position of the play operation. If this flag is not set, playback continues to the end of the media or segment, as defined by the device. If the to position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_PLAY_SPEED
: This flag adds a speed parameter. The units are specified by the currently set speed format. The speed value is in the *ulSpeed* field in the play data structure.

MCI_DGV_PLAY_REVERSE
: This flag specifies to play in reverse.

MCI_DGV_PLAY_FAST
: This flag specifies to play at the fast rate (twice the normal recorded playback rate).

MCI_DGV_PLAY_SCAN
: Specifies to scan. Scan usually means to play as quickly as possible, with audio disabled.

MCI_DGV_PLAY_SLOW
: This flag specifies to play at the slow rate (half the normal recorded playback rate).

MCI_DGV_PLAY_REPEAT
: This flag specifies that the play operation be repeated until the command is superseded by another command or aborted.

  This flag is not supported by the digital video MCD.

Videodisc Extensions

The following additional flags apply to videodisc devices. MCI_VD_PLAY_REVERSE and MCI_VD_PLAY_SCAN are mutually exclusive. Only one of the other flags is allowed with this message.

MCI_VD_PLAY_REVERSE
: This flag specifies to play in reverse.

MCI_VD_PLAY_FAST
: This flag specifies to play at the fast rate.

MCI_VD_PLAY_SCAN
: This flag specifies to scan. Scan usually means to play as fast as possible, with audio disabled.

MCI_VD_PLAY_SPEED
: This flag adds a speed parameter. The units are specified by the currently set speed format. The speed value is in the *ulSpeed* field of the play data structure.

MCI_VD_PLAY_SLOW
: This flag specifies to play at the slow rate.

pParam2 (PMCI_PLAY_PARMS)
: A pointer to an MCI_PLAY_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

  PMCI_DGV_PLAY_PARMS
  : A pointer to an MCI_DGV_PLAY_PARMS data structure.

  PMCI_VD_PLAY_PARMS
  : A pointer to an MCI_VD_PLAY_PARMS data structure.

rc (ULONG)
: Return codes indicating success or type of failure:

  MCIERR_SUCCESS
  : If the function succeeds, 0 is returned.

MCIERR_MEDIA_CHANGED	The required media has changed.
MCIERR_DEVICE_NOT_READY	The device is not ready.
MCIERR_INVALID_DEVICE_ID	The device id is not VALID.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_UNSUPPORTED_FUNCTION	Unsupported function.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_OUTOFRANGE	Units are out of range.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_CHANNEL_OFF	Primary channel is off.

MCI_PLAY - Remarks

The parameters and flags for this message vary according to the selected device. The units of the MCI_FROM and MCI_TO parameters must be supplied in the currently selected time format. See the MCI_SET message and the MCI_SET_TIME_FORMAT flag for more information.

The following example illustrates how the MCI_FROM and MCI_TO parameters are interpreted. If a multimedia element is composed of samples; in a file with 100 samples, the samples are numbered from 0 to 99. If MCI_FROM is specified as 10 and MCI_TO is specified as 80, MCI_PLAY will play samples 10 through 79. Following the play operation, the current position of the media would be 80.

If the length of a file cannot be determined, MCIERR_SUCCESS might be returned even if the MCI_TO parameter is out of range.

Digital Video Specific

If you are using an application-defined window and your application is running on a system without direct-access device driver support for motion video, do *not* issue MCI_PLAY with the MCI_WAIT flag specified unless the thread issuing the message is separate from the thread reading the message queue.

MCI_PLAY - Default Processing

If MCI_FROM is not specified, the starting position defaults to the current location.

IF MCI_TO is not specified, playback continues to the end of the media or segment, as defined by the device.

MCI_PLAY - Related Messages

- MCI_RECORD
- MCI_PAUSE
- MCI_RESUME
- MCI_STOP

MCI_PLAY - Example Code

The following code illustrates how to play a device from 5 to 25 seconds with the time format set to milliseconds.

```
USHORT          usDeviceID;
MCI_PLAY_PARMS  mpp;

/* Play from 5 seconds to 25 seconds (time format set to
   milliseconds)                                              */

/* Assign hwndCallback the handle to the PM Window routine    */
mpp.hwndCallback = (HWND) hwndMyWindow;

mpp.ulFrom = (ULONG) 5000;       /* Play from this position   */
mpp.ulTo = (ULONG) 25000;        /* Play to this position     */

mciSendCommand(usDeviceID,       /* Device ID                 */
               MCI_PLAY,         /* MCI play message          */
               MCI_NOTIFY | MCI_FROM | MCI_TO,
                                 /* Flags for this message    */
               (PVOID) &mpp,     /* Data structure            */
               0);               /* No user parm              */
```

MCI_PLAY - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Related Messages
Example Code
Glossary

MCI_PUT

MCI_PUT Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices supporting MCI_PUT:

MCI_DGV_PUT_RECT
This flag specifies that the *rc* field of the data structure identified by *pParam2* contains a valid display rectangle array. This is a required parameter.

MCI_DGV_PUT_SOURCE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle array specifying the offset and size of a clipping rectangle for the digital video source image. The source rectangle array specifies a capture rectangle relative to the digital video origin. MCI_DGV_PUT_SOURCE is only valid with the MCI_DGV_RECORD flag.

Note: The size of the origin (or source) can be found using MCI_DGV_STATUS_VIDEO_X_EXTENT and MCI_DGV_STATUS_VIDEO_Y_EXTENT.

MCI_DGV_PUT_DESTINATION
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle array specifying the offset and visible extent of the digital video within the client window. The destination rectangle array specifies a clipping rectangle for frames relative to the lower-left corner of the window. When MCI_DGV_PUT_DESTINATION is used with MCI_DGV_RECORD, the size of the movie to be recorded is determined and the position is ignored. When MCI_DGV_PUT_DESTINATION is used with MCI_DGV_MONITOR, the size and position of the monitor video relative to the monitor window is determined. If MCI_DGV_PUT_DESTINATION is used without either MCI_DGV_MONITOR or MCI_DGV_RECORD, the size and position of the playback video relative to the playback window is determined.

MCI_DGV_PUT_WINDOW_MOVE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle specifying the window position. All four values (X1 Y1 X2 Y2) must be specified, but X2 and Y2 are ignored unless the MCI_DGV_PUT_WINDOW_SIZE parameter is also specified.

MCI_DGV_PUT_WINDOW_SIZE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle that specifies the size of the window. All four values (X1 Y1 X2 Y2) must be specified.

MCI_DGV_RECORD
Specifies the source and destination rectangles for video capture.

Note: For recording, the source rectangle specifies the portion of the image to be captured and the destination rectangle specifies the size of the video to be recorded, thereby indicating the scaling to be applied to the source rectangle.

MCI_DGV_MONITOR
This flag specifies the window size and position for the monitor window.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_PUT_RECT
Specifies that the *rc* field of the data structure identified by *pParam2* contains a valid display rectangle.

MCI_OVLY_PUT_DESTINATION
: Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle for the video overlay within the client window. The destination rectangle specifies a clipping rectangle for frames relative to the lower-left corner of the window. If MCI_OVLY_PUT_DESTINATION is specified without the MCI_OVLY_PUT_RECT flag specified, the default destination is set.

MCI_OVLY_PUT_SOURCE
: Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle for the analog video source. The source rectangle specifies the portion of the incoming video signal which will be displayed. If MCI_OVLY_PUT_SOURCE is specified without the MCI_OVLY_PUT_RECT flag specified, the default source is set.

MCI_OVLY_PUT_WINDOW_MOVE
: Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle, where the X1 Y1 coordinates specify the new location of the default video window. The coordinates are relative to the parent window. The X2 and Y2 coordinates are ignored unless the MCI_OVLY_PUT_WINDOW_SIZE flag is also specified.

MCI_OVLY_PUT_WINDOW_SIZE
: Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle. The new default window size is calculated to ((X2 - X1) + 1) and ((Y2 - Y1) + 1).

MCI_PUT Parameter - pParam2

pParam2 (PMCI_VID_RECT_PARMS)
A pointer to the MCI_VID_RECT_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_RECT_PARMS
: A pointer to the MCI_DGV_RECT_PARMS data structure.

PMCI_OVLY_RECT_PARMS
: A pointer to the MCI_OVLY_RECT_PARMS data structure.

MCI_PUT Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_MISSING_FLAG

Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_PUT - Description

This message sets the source and destination rectangles for the transformation of the video image and the position of the default video window.

ulParam1 (ULONG)
This parameter can contain any of the following:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices supporting MCI_PUT:

MCI_DGV_PUT_RECT
This flag specifies that the *rc* field of the data structure identified by *pParam2* contains a valid display rectangle array. This is a required parameter.

MCI_DGV_PUT_SOURCE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle array specifying the offset and size of a clipping rectangle for the digital video source image. The source rectangle array specifies a capture rectangle relative to the digital video origin. MCI_DGV_PUT_SOURCE is only valid with the MCI_DGV_RECORD flag.

Note: The size of the origin (or source) can be found using MCI_DGV_STATUS_VIDEO_X_EXTENT and MCI_DGV_STATUS_VIDEO_Y_EXTENT.

MCI_DGV_PUT_DESTINATION
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle array specifying the offset and visible extent of the digital video within the client window. The destination rectangle array specifies a clipping rectangle for frames relative to the lower-left corner of the window. When MCI_DGV_PUT_DESTINATION is used with MCI_DGV_RECORD, the size of the movie to be recorded is determined and the position is ignored. When MCI_DGV_PUT_DESTINATION is used with MCI_DGV_MONITOR, the size and position of the monitor video relative to the monitor window is determined. If MCI_DGV_PUT_DESTINATION is used without either MCI_DGV_MONITOR or MCI_DGV_RECORD, the size and position of the playback video relative to the playback window is determined.

MCI_DGV_PUT_WINDOW_MOVE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle specifying the window position. All four values (X1 Y1 X2 Y2) must be specified, but X2 and Y2 are ignored unless the MCI_DGV_PUT_WINDOW_SIZE parameter is also specified.

MCI_DGV_PUT_WINDOW_SIZE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle that specifies the size of the window. All four values (X1 Y1 X2 Y2) must be specified.

MCI_DGV_RECORD
Specifies the source and destination rectangles for video capture.

Note: For recording, the source rectangle specifies the portion of the image to be captured and the destination rectangle specifies the size of the video to be recorded, thereby indicating the scaling to be applied to the source rectangle.

MCI_DGV_MONITOR
This flag specifies the window size and position for the monitor window.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_PUT_RECT
Specifies that the *rc* field of the data structure identified by *pParam2* contains a valid display rectangle.

MCI_OVLY_PUT_DESTINATION
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle for the video overlay within the client window. The destination rectangle specifies a clipping rectangle for frames relative to the lower-left corner of the window. If MCI_OVLY_PUT_DESTINATION is specified without the MCI_OVLY_PUT_RECT flag specified, the default destination is set.

MCI_OVLY_PUT_SOURCE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle for the analog video source. The source rectangle specifies the portion of the incoming video signal which will be displayed. If MCI_OVLY_PUT_SOURCE is specified without the MCI_OVLY_PUT_RECT flag specified, the default source is set.

MCI_OVLY_PUT_WINDOW_MOVE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle, where the X1 Y1 coordinates specify the new location of the default video window. The coordinates are relative to the parent window. The X2 and Y2 coordinates are ignored unless the MCI_OVLY_PUT_WINDOW_SIZE flag is also specified.

MCI_OVLY_PUT_WINDOW_SIZE
Indicates that the *rc* field of the data structure identified by *pParam2* contains a display rectangle. The new default window size is calculated to ((X2 - X1) + 1) and ((Y2 - Y1) + 1).

pParam2 (PMCI_VID_RECT_PARMS)
A pointer to the MCI_VID_RECT_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_RECT_PARMS
A pointer to the MCI_DGV_RECT_PARMS data structure.

PMCI_OVLY_RECT_PARMS
A pointer to the MCI_OVLY_RECT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_PUT - Remarks

Not all devices support distorting the source rectangle image to fit the display rectangle. The MCI_GETDEVCAPS message (MCI_DGV_GETDEVCAPS_CAN_DISTORT) can be used to determine whether the device supports distorting.

If either the width or the height of the rectangle specified with MCI_DGV_PUT_DESTINATION and MCI_DGV_RECORD is not a multiple of eight, then that value is rounded to the nearest multiple of eight. If the device cannot distort and the rectangle specified with MCI_DGV_PUT_SOURCE and MCI_DGV_RECORD is not an integral multiple of the rectangle specified with MCI_DGV_PUT_DESTINATION and MCI_DGV_RECORD, the source and destination rectangles are adjusted to find the nearest values that will make the source be an integral multiple of the destination and the destination be a multiple of eight.

When the device is monitoring while recording or monitoring while cued for input, the video seen in the monitor window will be the content in the record source rectangle set with MCI_DGV_PUT_SOURCE and MCI_DGV_RECORD. When the device is monitoring while not recording or cued for input, the video seen in the monitor window will be the maximum source (full video extent of the capture card reported by MCI_DGV_STATUS_X_EXTENT and MCI_DGV_STATUS_Y_EXTENT), and an animated, dashed-line rectangle will be drawn on the monitor window to indicate the relative position of the record source rectangle.

If both window move and size flags are specified, then all four window coordinates must be provided.

An application-supplied alternate video window will *not* be affected by the window move or size flags.

MCI_PUT - Related Messages

- MCI_WINDOW

MCI_PUT - Example Code

The following code illustrates how to set the source and destination rectangle arrays for the transformation of the video.

```
MCI_DGV_RECT_PARMS mciRectParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;

/* An example of changing the SOURCE area to a
   sub-rectangle of the total input */
memset (&mciRectParms, 0x00, sizeof (MCI_DGV_RECT_PARMS));
mciRectParms.hwndCallback = hwndNotify;
mciRectParms.rc.xLeft    = lX1;
mciRectParms.rc.yBottom = lY1;
mciRectParms.rc.xRight  = lX2;
mciRectParms.rc.yTop    = lY2;


ulReturn = mciSendCommand(usDeviceID, MCI_PUT,
               MCI_WAIT | MCI_DGV_PUT_RECT |
               MCI_DGV_RECORD | MCI_DGV_PUT_SOURCE,
               (PVOID)&mciRectParms,
               usUserParm);
```

MCI_PUT - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_RECORD

MCI_RECORD Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
Indicates a starting position is included in the *ulFrom* field of the data structure pointed to by *pParam2*. The units assigned to the position values are specified with the MCI_SET_TIME_FORMAT flag of the MCI_SET command. If MCI_FROM is not specified, the starting position defaults to the current location. The *ulFrom* field refers to a position in the destination media.

MCI_TO
Indicates an ending position is included in the *ulTo* field of the data structure pointed to by *pParam2*. The units assigned to the position values are specified with the MCI_SET_TIME_FORMAT flag of the MCI_SET command. If MCI_TO is not specified, the record will continue until a pause or stop message is received. The *ulTo* field refers to a position in the destination media.

MCI_RECORD_INSERT
Indicates that newly recorded information is to be inserted into existing data at the current location. Some devices, such as non-file-oriented devices, do not support this.

MCI_RECORD_OVERWRITE
Indicates that recorded data is to overwrite existing data at the current location. Note that MCI_RECORD_INSERT and MCI_RECORD_OVERWRITE are mutually exclusive.

MCI_RECORD Parameter - pParam2

pParam2 (PMCI_RECORD_PARMS)
A pointer to the MCI_RECORD_PARMS data structure.

MCI_RECORD Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_OUTOFRANGE
The value supplied in the *ulFrom* field of the data structure identified by *pParam2* is greater than the size of the element.

MCIERR_OUT_OF_MEMORY
There is insufficient memory to complete the requested action.

MCIERR_TARGET_DEVICE_FULL
The target device is full.

MCI_RECORD - Description

This message causes the device to start recording. Before you send this message, it is recommended that you issue MCI_ACQUIREDEVICE with the MCI_EXCLUSIVE_INSTANCE flag set. This will lock the device context and prevent it from being made inactive.

Digital Video Specific

This message initiates real-time recording of motion video with simultaneous audio capture. Any options, such as frame rate, quality, and so on, in effect at the time recording starts are applied to the recording and cannot be changed during the recording process. If changes to recording options or parameters are attempted during recording, MCIERR_INVALID_MODE is returned. All recording operations entirely replace the contents of the device element at the starting location. MCI_FROM is not supported and MCI_TO is used only as an indication of the length of the recording to be performed.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_FROM
Indicates a starting position is included in the *ulFrom* field of the data structure pointed to by *pParam2*. The units assigned to the position values are specified with the MCI_SET_TIME_FORMAT flag of the MCI_SET command. If MCI_FROM is not specified, the starting position defaults to the current location. The *ulFrom* field refers to a position in the destination media.

MCI_TO
Indicates an ending position is included in the *ulTo* field of the data structure pointed to by *pParam2*. The units assigned to the position values are specified with the MCI_SET_TIME_FORMAT flag of the MCI_SET command. If MCI_TO is not specified, the record will continue until a pause or stop message is received. The *ulTo* field refers to a position in the destination media.

MCI_RECORD_INSERT
Indicates that newly recorded information is to be inserted into existing data at the current location. Some devices, such as non-file-oriented devices, do not support this.

MCI_RECORD_OVERWRITE
Indicates that recorded data is to overwrite existing data at the current location. Note that MCI_RECORD_INSERT and MCI_RECORD_OVERWRITE are mutually exclusive.

pParam2 (PMCI_RECORD_PARMS)
A pointer to the MCI_RECORD_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_FILE_NOT_FOUND	File has not been loaded.
MCIERR_MISSING_PARAMETER	Required parameter is missing.
MCIERR_OUTOFRANGE	The value supplied in the *ulFrom* field of the data structure identified by *pParam2* is greater than the size of the element.
MCIERR_OUT_OF_MEMORY	There is insufficient memory to complete the requested action.
MCIERR_TARGET_DEVICE_FULL	The target device is full.

MCI_RECORD - Remarks

The units of the MCI_FROM and MCI_TO parameters must be supplied in the currently selected time format. See the MCI_SET message and the MCI_SET_TIME_FORMAT flag for more information.

Only devices that return TRUE to the MCI_GETDEVCAPS_CAN_RECORD flag of the MCI_GETDEVCAPS command support this message.

A STOP is performed implicitly if the device is not stopped when MCI_RECORD is issued. If a STOP is issued during recording, MCI_NOTIFY_ABORTED will be returned. If an MCI_TO position is specified on a record operation and the record operation completes, MCI_NOTIFY_SUCCESSFUL is returned.

MCI_RECORD - Default Processing

If MCI_FROM is not specified, the starting position defaults to the current location.

If MCI_TO is not specified, the record continues until a pause or stop message is received.

MCI_RECORD_INSERT is the default for devices that support insert. MCI_RECORD_OVERWRITE is the default for devices that do not support insert.

Waveaudio Specific

Although insert is supported by the waveaudio device, the default is overwrite for recording operations.

MCI_RECORD - Related Messages

- MCI_PAUSE
- MCI_RESUME
- MCI_SAVE
- MCI_STOP

MCI_RECORD - Example Code

The following code illustrates how to start recording at the 5 second position in the current device element, and then overwrite existing data by recording for 5 seconds.

```
USHORT            usDeviceID;
MCI_RECORD_PARMS  mrp;

 /* Start recording at the 5 second position in the current device
    element, and record for 5 seconds, overwriting existing data.     */

 /* Assumes time format set to milliseconds                           */

mrp.hwndCallback = hwndMyWindow;
                 /* Assign hwndCallback the handle to the PM Window   */
mrp.ulFrom = (ULONG)  5000;                 /* Record from position   */
mrp.ulTo = (ULONG) 10000;                   /* Record to position     */

mciSendCommand(usDeviceID,                  /* Device ID              */
               MCI_RECORD,                  /* MCI record message     */
               MCI_NOTIFY | MCI_FROM |
               MCI_TO |MCI_RECORD_OVERWRITE,
                                            /* Flags for this message */
               (ULONG) &mrp,                /* Data structure         */
               0);                          /* No user parm           */
```

MCI_RECORD - Topics

Select an item:

MCI_REDO

MCI_REDO Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_REDO Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_REDO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	Redo was successful.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_CANNOT_REDO	Redo is not possible in the current state.

MCI_REDO - Description

This message redoes the cut, paste, or delete operation most recently undone by MCI_UNDO. MCI_REDO should immediately follow MCI_UNDO; otherwise, editing actions performed after MCI_UNDO (and before a corresponding MCI_REDO) will be lost when MCI_REDO is issued. The media position is at the beginning of the file after a redo operation.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	Redo was successful.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_CANNOT_REDO	Redo is not possible in the current state.

MCI_REDO - Remarks

MCI_REDO operates on one editing action (for example, cut, delete, paste) at a time. If there are no more possible actions to be redone (that is, the file is in the state where the last change was made), then MCIERR_CANNOT_REDO is returned.

Note: The redo operation is unlimited corresponding to the number of undo operations that have been performed. However, after a save, any previous editing actions are cleared and cannot be redone.

Not all devices support this message. To determine if a device supports MCI_REDO, issue MCI_GETDEVCAPS.

If the redo operation interrupts an in-progress operation, such as play, the command is aborted and an MM_MCINOTIFY message will be sent to the application.

MCI_REDO - Related Messages

- MCI_COPY
- MCI_CUT
- MCI_PASTE
- MCI_DELETE
- MCI_UNDO

MCI_REDO - Example Code

The following code illustrates redoing the last editing action most recently undone.

```
  USHORT                usDeviceID;
  MCI_EDIT_PARMS        mep;

  mep.hwndCallback = hwndMyWindow;
```

```
    mciSendCommand(usDeviceID,
                   MCI_REDO,
                   MCI_NOTIFY,
                   &mep,
                   0 );
```

MCI_REDO - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_RELEASEDEVICE

MCI_RELEASEDEVICE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_RETURN_RESOURCE
This flag releases a device instance from the active state and makes the next available inactive device instance active. The device instance will not be made active again unless MCI_ACQUIREDEVICE is issued for this device instance, or no other application is using the device. If the instance is already inactive, the message is ignored.

MCI_RELEASEDEVICE Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_RELEASEDEVICE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_RELEASEDEVICE - Description

This message is sent to release the exclusive use of physical device resources by a group or device instance.

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_RETURN_RESOURCE
This flag releases a device instance from the active state and makes the next available inactive device instance active. The device instance will not be made active again unless MCI_ACQUIREDEVICE is issued for this device instance, or no other application is using the device. If the instance is already inactive, the message is ignored.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_RELEASEDEVICE - Remarks

Releasing a device does not always cause the device to be passed to another application. Ownership of a device is changed only when the MCI_ACQUIREDEVICE message is used, or if another application closes or opens a device.

MCI_RELEASEDEVICE - Example Code

The following code illustrates how to acquire and then release a device.

```
MCI_GENERIC_PARMS mciGenericParms;
                                      /* Info data structure for cmd */
USHORT   usDeviceID;
HWND     hwndMyWindow;

         /* Assign hwndCallback the handle to the PM Window routine */

mciGenericParms.hwndCallback = hwndMyWindow;


 /* Acquire the device for exclusive access and assume it is inactive */

mciSendCommand(usDeviceID,            /* Device ID                   */
     MCI_ACQUIREDEVICE,               /* MCI acquire device message  */
     MCI_NOTIFY |  MCI_EXCLUSIVE,
                                      /* Flags for this message      */
     (PVOID) &mciGenericParms,        /* Data structure              */
     0);                              /* No user parm                */

   /* Device will be exclusively acquired once MM_MCIPASSDEVICE      */
      message is received indicating MCI_GAINING_USE                 */

   /* Perform whatever operations require exclusive access to device */

 mciSendCommand(usDeviceID,           /* Device ID                   */
   MCI_RELEASEDEVICE,                 /* MCI release device message  */
   MCI_NOTIFY,                        /* Flag for this message       */
   (PVOID) &mciGenericParms,          /* Data structure              */
   0);                                /* No user parm                */
```

MCI_RELEASEDEVICE - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_RESTORE

MCI_RESTORE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_RESTORE_SRC_RECT
The *SrcRect* field of the MCI_RESTORE_PARMS data structure contains a rectangle specifying the area to be restored from the capture device element. If this flag is not specified, the entire image is restored.

MCI_RESTORE_DEST_RECT
The *DestRect* field of the MCI_RESTORE_PARMS data structure contains a rectangle specifying the destination area of the window to be restored. If this flag is not specified, the destination size is assumed to be the same as the image size in device coordinates placed at the lower-left corner of the window.

MCI_RESTORE Parameter - pParam2

pParam2 (PMCI_RESTORE_PARMS)
A pointer to an MCI_RESTORE_PARMS data structure.

MCI_RESTORE Return Value - rc

rc (ULONG)
This function fails if nothing is currently in the capture device element.

Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM/2 driver for this command.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available. (For example, video has been set off.)

MCI_RESTORE - Description

This message causes a video device to transfer an image from the element buffer to the display surface. To ensure that the image is displayed, the device automatically performs a freeze operation where necessary.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_RESTORE_SRC_RECT
The *SrcRect* field of the MCI_RESTORE_PARMS data structure contains a rectangle specifying the area to be restored from the capture device element. If this flag is not specified, the entire image is restored.

MCI_RESTORE_DEST_RECT
The *DestRect* field of the MCI_RESTORE_PARMS data structure contains a rectangle specifying the destination area of the window to be restored. If this flag is not specified, the destination size is assumed to be the same as the image size in device coordinates placed at the lower-left corner of the window.

pParam2 (PMCI_RESTORE_PARMS)
A pointer to an MCI_RESTORE_PARMS data structure.

rc (ULONG)
This function fails if nothing is currently in the capture device element.

Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM/2 driver for this command.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available. (For example, video has been set off.)

MCI_RESTORE - Remarks

The image is restored from the device element in an overlay video device. It is also restored from the still image device element of a digital video device.

In the case of overlay video and digital video devices implemented on dual-plane video hardware, the image is restored to the *video* or *image* layer.

Devices capable of scaling the image will attempt to do so in order to transform the output to the destination rectangle. If a destination rectangle is not specified or the device is not capable of scaling the image, the output is clipped to the destination rectangle as required.

MCI_RESTORE - Example Code

The following code illustrates how to cause a video device to transfer an image from the image device element buffer to the display surface.

```
MCI_IMAGE_PARMS mciImageParms;
MCI_RESTORE_PARMS mciRestoreParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;

/* Without a rectangle */
memset (&mciRestoreParms, 0x00, sizeof (MCI_RESTORE_PARMS));
mciRestoreParms.hwndCallback = hwndNotify;
mciRestoreParms.DestRect   = 0;

ulReturn = mciSendCommand(usDeviceID, MCI_RESTORE,
               MCI_WAIT,
               (PVOID)&mciRestoreParms,
               usUserParm);

/* With a rectangle */
memset (&mciRestoreParms, 0x00, sizeof (MCI_RESTORE_PARMS));
mciRestoreParms.hwndCallback = hwndNotify;
mciRestoreParms.DestRect.xLeft   = lX1;
mciRestoreParms.DestRect.yBottom = lY1;
mciRestoreParms.DestRect.xRight  = lX2;
mciRestoreParms.DestRect.yTop    = lY2;

ulReturn = mciSendCommand(usDeviceID, MCI_RESTORE,
               MCI_WAIT | MCI_RESTORE_DEST_RECT,
               (PVOID)&mciRestoreParms,
               usUserParm);
```

MCI_RESTORE - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_RESUME

MCI_RESUME Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_RESUME Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_RESUME Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device context is not valid.

MCIERR_INSTANCE_INACTIVE
The device context is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_UNSUPPORTED_FLAG

Specified flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Specified callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_RESUME - Description

This message is sent to resume playing or recording from a paused state.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device context is not valid.

MCIERR_INSTANCE_INACTIVE
The device context is currently inactive. Issue MCI_ACQUIREDEVICE to make device context active.

MCIERR_UNSUPPORTED_FLAG
Specified flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Specified callback handle is invalid.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_RESUME - Remarks

The previously specified **to** parameter remains in effect.

MCI_RESUME - Related Messages

- MCI_RECORD
- MCI_PAUSE
- MCI_PLAY

MCI_RESUME - Example Code

The following code illustrates how to resume a paused operation.

```
USHORT             usDeviceID;
HWND               hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms;          /* Generic message
                                               parms structure       */

 /* Resume the previous operation that was paused                    */

 /* Assign hwndCallback the handle to the PM Window routine          */
mciGenericParms.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,                 /* Device ID             */
                MCI_RESUME,                 /* MCI resume message    */
                MCI_NOTIFY,                 /* Flag for this message */
                (PVOID) &mciGenericParms,   /* Data structure        */
                0);                         /* No user parm          */
```

MCI_RESUME - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_REWIND

MCI_REWIND Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following standard flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_REWIND Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_REWIND Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_DEVICE_LOCKED	The device is acquired for exclusive use.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_INVALID_CALLBACK_HANDLE	The callback handle given is not correct.

MCI_REWIND - Description

This message seeks the media to the starting position. This position is defined as the first "playable" area, beyond any header or table-of-contents data.

ulParam1 (ULONG)
This parameter can contain any of the following standard flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_DEVICE_LOCKED
The device is acquired for exclusive use.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_CALLBACK_HANDLE
The callback handle given is not correct.

MCI_REWIND - Remarks

This message is the equivalent of the MCI_SEEK message with the MCI_TO_START flag specified.

MCI_REWIND - Example Code

The following code illustrates how to seek the media to the starting position.

```
USHORT            usDeviceID;
HWND              hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms;
                          /* Generic message parms structure */

/* Assign hwndCallback the handle to the PM Window routine */
mciGenericParms.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,               /* Device ID             */
                MCI_REWIND,               /* MCI rewind message    */
                MCI_NOTIFY,               /* Flag for this message */
                (PVOID) &mciGenericParms, /* Data structure        */
                0);                       /* No user parm          */
```

MCI_REWIND - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_SAVE

MCI_SAVE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SAVE_FILE
The *pszFileName* field of the MCI_SAVE_PARMS data structure contains the destination file name. If a file name is not specified, the original file opened or the most recently loaded file name is assumed.

Digital Video Extensions

The following additional flags apply to digital video devices.

MCI_DGV_SAVE_VIDEO_FILE
Saves the motion video device element.

MCI_DGV_SAVE_IMAGE_FILE
Saves the still image device element.

MCI_SAVE Parameter - pParam2

pParam2 (PMCI_SAVE_PARMS)
A pointer to the MCI_SAVE_PARMS data structure.

MCI_SAVE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_TARGET_DEVICE_FULL	Target device is full.
MCIERR_FILE_NOT_FOUND	File not found.
MCIERR_FILE_NOT_SAVED	File not saved.
MCIERR_FILE_ATTRIBUTE	File attribute error.
MMIOERR_NEED_NEW_FILE_NAME	The file cannot be saved with its original name because there are other processes that have outstanding paste operations using the data in this file. Saving the file with its original name will cause this data to be lost.
MMIOERR_CLIPBRD_ACTIVE	The file cannot be saved with its original name because there is an active reference to its data in the clipboard. Saving the file with its original name will cause this data to be lost.

MCI_SAVE - Description

This message saves the current file.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SAVE_FILE
The *pszFileName* field of the MCI_SAVE_PARMS data structure contains the destination file name. If a file name is not specified, the original file opened or the most recently loaded file name is assumed.

Digital Video Extensions

The following additional flags apply to digital video devices.

MCI_DGV_SAVE_VIDEO_FILE
Saves the motion video device element.

MCI_DGV_SAVE_IMAGE_FILE
Saves the still image device element.

pParam2 (PMCI_SAVE_PARMS)
A pointer to the MCI_SAVE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_TARGET_DEVICE_FULL
Target device is full.

MCIERR_FILE_NOT_FOUND
File not found.

MCIERR_FILE_NOT_SAVED
File not saved.

MCIERR_FILE_ATTRIBUTE
File attribute error.

MMIOERR_NEED_NEW_FILE_NAME
The file cannot be saved with its original name because there are other processes that have outstanding paste operations using the data in this file. Saving the file with its original name will cause this data to be lost.

MMIOERR_CLIPBRD_ACTIVE
The file cannot be saved with its original name because there is an active reference to its data in the clipboard. Saving the file with its original name will cause this data to be lost.

MCI_SAVE - Remarks

If the MCI_SAVE_FILE flag is specified, the current device element is saved with the file name specified in the *pszFileName* field. The file specified in *pszFileName* becomes the currently loaded element. If the MCI_SAVE_FILE flag is not specified or if *pszFileName* is NULL, MCI_SAVE saves to the currently loaded element name of the device instance. If the current element has not been named, MCIERR_FILE_NOT_FOUND is returned.

This command is supported by devices which return TRUE to the MCI_GETDEVCAPS_CAN_SAVE query using the MCI_GETDEVCAPS message.

The IBM sequencer device does not currently support this message.

Digital Video Specific

The MCI_DGV_SAVE_VIDEO_FILE flag is not required; saving a video file is assumed by default. An edited AVI movie file cannot always be saved with its original name. If the clipboard contains a reference to data that would be erased during saving or if another instance of the digital video device has a pending paste operation that depends on this data, the file cannot be saved unless a new file name is provided. If a new file name is not provided, the MMIOERR_NEED_NEW_FILENAME error is returned by the AVI I/O procedure and a temporary file is created to save the edited movie. The AVI I/O procedure alerts the user by displaying a message with the name of the temporary file that was created. The application must reopen the temporary file to use the edited version of the movie.

During setup for MMIOM_SAVE processing, the AVI I/O procedure checks to see if the clipboard contains data from a file and if the file needs to be rewritten. If these conditions are true, the save operation is aborted and the MMIOERR_CLIPBRD_ACTIVE error is returned.

MCI_SAVE - Related Messages

- MCI_LOAD
- MCI_OPEN
- MCI_RECORD

MCI_SAVE - Example Code

The following code illustrates how to save a device element to a new file and receive notification upon completion.

```
USHORT                usDeviceID;
HWND                  hwndMyWindow;
MCI_SAVE_PARMS        msp;

/* Assign hwndCallback the handle to the PM Window                    */

msp.hwndCallback = hwndMyWindow;


msp.pszFileName = (PVOID) "movie.avi";      /* File name to save      */

mciSendCommand( usDeviceID,                 /* Device ID              */
  MCI_SAVE,                                 /* MCI save message       */
  MCI_NOTIFY | MCI_SAVE_VIDEO_FILE,
                                            /* Flags for this message */
```

```
   (PVOID) &msp,                          /* Data structure          */
   0);                                    /* No user parm            */
```

MCI_SAVE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_SEEK

MCI_SEEK Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_TO
This flag indicates that the *ulTo* field of the MCI_SEEK_PARMS data structure specifies the ending position of the seek operation. If the *ulTo* position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

MCI_TO_START
This flag causes the device to seek to the first playable position on the media. This is not necessarily position 0.

MCI_TO_END
This flag causes the device to seek to the end of the media.

Digital Video Extensions

The following additional flag applies to digital video drivers.

MCI_TO_NEAREST_IFRAME
This flag causes the device to seek to the nearest I-frame preceding the point specified by MCI_TO.

Videodisc Extensions

The following additional flag applies to videodisc device drivers.

MCI_VD_SEEK_REVERSE

This flag initiates a seek backward.

MCI_SEEK Parameter - pParam2

pParam2 (PMCI_SEEK_PARMS)
A pointer to the MCI_SEEK_PARMS structure.

MCI_SEEK Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INSTANCE_INACTIVE	The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_HARDWARE	Device hardware error.
MCIERR_UNSUPPORTED_FUNCTION	Unsupported function.
MCIERR_INVALID_FLAG	Flag (*ulParam1*) is invalid.
MCIERR_FLAGS_NOT_COMPATIBLE	Flags cannot be used together.
MCIERR_FILE_NOT_FOUND	File has not been loaded.
MCIERR_MISSING_PARAMETER	Required parameter is missing.

MCI_SEEK - Description

This message is sent to change the current media position of the device.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_TO
This flag indicates that the *ulTo* field of the MCI_SEEK_PARMS data structure specifies the ending position of the seek operation. If the *ulTo* position is beyond the end of the media or segment, an MCIERR_OUTOFRANGE error is returned.

MCI_TO_START
This flag causes the device to seek to the first playable position on the media. This is not necessarily position 0.

MCI_TO_END
This flag causes the device to seek to the end of the media.

Digital Video Extensions

The following additional flag applies to digital video drivers.

MCI_TO_NEAREST_IFRAME
This flag causes the device to seek to the nearest I-frame preceding the point specified by MCI_TO.

Videodisc Extensions

The following additional flag applies to videodisc device drivers.

MCI_VD_SEEK_REVERSE
This flag initiates a seek backward.

pParam2 (PMCI_SEEK_PARMS)
A pointer to the MCI_SEEK_PARMS structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SEEK - Remarks

The parameters and flags for this message vary according to the selected device. The values of the MCI_TO parameters must be specified in the currently selected time format. See the MCI_SET message and the MCI_SET_TIME_FORMAT flag for more information.

The following example illustrates how the MCI_TO parameter is interpreted. If a multimedia element is composed of samples; in a file with 100 samples, the samples are numbered from 0 to 99. If MCI_TO is specified as 0, the media is positioned at its start. If an MCI_PLAY message is issued, the first sample would be the first to play. If MCI_TO is specified as 99, the media is positioned before the last sample. Issuing an MCI_PLAY message would play the last sample. Specifying MCI_TO_END would position the media at the end of the file and the current position would be 100. At this point, if an MCI_PLAY message is issued, the command would return successfully without performing any operation.

MCI_SEEK - Related Messages

- MCI_SET

MCI_SEEK - Example Code

The following code illustrates how to seek to the beginning of the playable media for a device. Note that this might not be zero for all device types.

```
USHORT           usDeviceID;
MCI_SEEK_PARMS   mseekp;

/* Seek the device to the beginning                               */

/* Assign hwndCallback the handle to the PM Window                */
mseekp.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,              /* Device ID             */
 MCI_SEEK,                               /* MCI seek message      */
 MCI_NOTIFY | MCI_TO_START,              /* Flags for this message */
 (PVOID) &mseekp,                        /* Data structure        */
 0);                                     /* No user parm          */
```

MCI_SEEK - Topics

Select an item:

MCI_SET

MCI_SET Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_AUDIO
Sets audio attributes of the device instance. A device with audio capabilities might support both left and right channels. The channel is specified in the *ulAudio* field of the data structure specified by *pParam2*. The action to be taken is specified with the flags MCI_SET_ON (which enables audio output at the current volume level), MCI_SET_OFF (which mutes audio output), or MCI_SET_VOLUME. Specifying MCI_SET_VOLUME does not enable audio output if MCI_SET_OFF has been previously specified.

The following constants are defined for specifying the audio channel in the *ulAudio* field.

MCI_SET_AUDIO_ALL
Apply to both channels.

MCI_SET_AUDIO_LEFT
Apply to the left channel only.

MCI_SET_AUDIO_RIGHT
Apply to the right channel only.

MCI_SET_DOOR_OPEN
Instructs the device to open the media cover (if any). This message ejects the media from devices where appropriate.

MCI_SET_DOOR_CLOSED
Instructs the device to close the media cover (if any).

MCI_SET_DOOR_LOCK
Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

MCI_SET_DOOR_UNLOCK
Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

MCI_SET_VOLUME
Sets the level of audio as a percentage of the maximum audio level as indicated in the *ulLevel* field. The volume level that can be set on the device might be of coarser granularity than that specified. In this case, the actual level can be obtained by issuing a MCI_STATUS message. If a number greater than 100 is given, then 100 will be used as the volume setting, and no error will be returned. See Examples section for an example using this flag.

MCI_SET_VIDEO
: Sets the video signal on or off. This flag must be used with either MCI_SET_ON or MCI_SET_OFF.

MCI_SET_ON
: Sets the video or specified audio channel on.

MCI_SET_OFF
: Sets the video or specified audio channel off.

MCI_SET_SPEED_FORMAT
: Specifies the speed format to be used on subsequent commands contained in the *ulSpeedFormat* field. The following values can be used:

  MCI_FORMAT_PERCENTAGE
  : Specifies the subsequent speed values as a percentage of the normal speed.

  MCI_FORMAT_FPS
  : Specifies the subsequent speed values in frames per second. This is the default setting.

MCI_SET_TIME_FORMAT
: Uses a time format on subsequent commands. A time-format parameter must be indicated in the *ulTimeFormat* field of the data structure specified by *pParam2* if this flag is used. The default is MCI_FORMAT_MMTIME. The following time formats are generic; devices can also provide device-specific time units:

  MCI_FORMAT_MILLISECONDS
  : Indicates that all subsequent commands that specify time will do so in milliseconds for both input and output.

  MCI_FORMAT_MMTIME
  : Indicates that all subsequent commands that specify time will do so in MMTIME units for both input and output. This does not apply to command parameters that explicitly specify time units, such as milliseconds on *ulOver*.

MCI_OVER
: Sets the vectored delay time to change the volume (or other attribute) in milliseconds.

MCI_SET_ITEM
: Indicates that the item to be set is specified in the *ulItem* field of the data structure identified by *pParam2*. Any value associated with the item is contained in the *ulValue* field. Each item defines the use (if any) and meaning of the value in the *ulValue* field.

Amplifier Mixer Extensions

The following additional flags apply to amplifier-mixer devices. Only one audio attribute set function can be performed at a time with the MCI_SET message. The treble, bass, balance, pitch, and gain flags require the MCI_SET_AUDIO flag also to be set. The level to be set for each function is contained in the *ulLevel* field and represents a percentage of the maximum available audio effect provided by the device. Zero is the minimum effect, while 100 is the maximum effect.

The following audio effects apply to the final output mix. Any specification of a particular channel will be ignored.

MCI_AMP_SET_BALANCE
: Sets the final output balance. Zero is defined as full left balance while one hundred is defined as full right balance.

MCI_AMP_SET_BASS
: Controls bass as a percentage of the maximum achievable effect.

MCI_AMP_SET_GAIN
: Sets the gain as a percentage of the maximum achievable effect.

MCI_AMP_SET_PITCH
: Sets the pitch as a percentage of the maximum achievable effect.

MCI_AMP_SET_TREBLE
: Controls treble as a percentage of the maximum achievable effect.

The following items can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_AMP_SET_AUDIO
: Used with the extended ampmix audio attribute flags.

MCI_AMP_SET_MONITOR

Used with the MCI_SET_ON or MCI_SET_OFF flags. It instructs the ampmix device to monitor the currently selected connector. This flag is typically used to listen to (monitor) a source while it is being recorded by another device.

If the MCI_SET_ITEM flag is set and MCI_AMP_SET_AUDIO is in the *ulItem* field of MCI_AMP_SET_PARMS, the connector specified in *ulValue* can be modified with the following audio attribute flags in *ulAudio* and the appropriate level in *ulLevel*.

MCI_AMP_SET_ALC
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the auto-level control setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_BALANCE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the balance setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_BASS
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the bass setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CHORUS
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the chorus setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CROSSOVER
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the crossover setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM1
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100). for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM2
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM3
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_GAIN
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the gain setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_LOUDNESS
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the loudness setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MID
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the mid setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MONITOR
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the monitor setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MUTE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the mute setting for the connector specified in *ulValue*.

MCI_AMP_SET_PITCH
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the pitch setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_REVERB
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the reverb setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_STEREOENHANCE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the stereo enhance setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_TREBLE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the treble setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_VOLUME

The *ulLevel* field in MCI_AMP_SET_PARMS contains the volume setting as a percentage (0-100) for the connector specified in *ulValue* .

CD Audio Extensions

The following additional time formats are supported by CD audio devices and can be specified as values for the *ulTimeFormat* field of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_MSF

Indicates that all subsequent commands that specify time will do so in *mm:ss:ff* where *mm* is minutes, *ss* is seconds and *ff* is frames.

MCI_FORMAT_TMSF

Indicates that all subsequent commands that specify time will do so in *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and ff is frames.

CD/XA Extensions

The following additional flags apply to the CD/XA device. Only one channel set function can be performed at a time with the MCI_SET message. The channel is specified in the *ulChannel* field of the data structure. The destination of the data in that channel is determined by the flags below. Only one destination can be selected at a time with the MCI_SET message. This message must be used with the MCI_CDXA_SET_CHANNEL flag and either the MCI_SET_ON or MCI_SET_OFF flags.

MCI_CDXA_AUDIO_DEVICE

Sends the audio stream to the audio card.

MCI_CDXA_AUDIO_BUFFER

Sends the audio stream to a playlist.

MCI_CDXA_VIDEO_BUFFER

Sends the video stream to a playlist.

MCI_CDXA_DATA_BUFFER

Sends the data stream to a playlist.

Digital Video Extensions

The following additional items can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_DGV_SET_VIDEO_COMPRESSION

Specifies the FOURCC compression format used for recording digital motion video. The values that can be specified are:

MCI_VID_COMP_ULTI	Ultimotion
MCI_VID_COMP_DIB	Raw (uncompressed format)
MCI_VID_COMP_RT21	Indeo 2.1
MCI_VID_COMP_IV31	Indeo 3.1

The default compression type is specified through the Setup page for the digital video device. The initial setting is MCI_VID_COMP_ULTI until changed in the Setup.

Note: Compressors are not available for FLIC, MPEG, and Indeo 3.2 in this version of OS/2.

MCI_DGV_SET_RECORD_AUDIO

Sets audio soundtrack recording on or off. The default is MCI_ON. This flag is used with MCI_ON or MCI_OFF.

MCI_DGV_SET_REF_INTERVAL

Sets the frequency at which reference frames (or I-frames) are to be compressed in the output data stream. A value of 0 results in no I-frames, a value of 1 causes every frame to be an I-frame, a value of 2 causes every other frame to be an I-frame, and so on. While there is no upper bound on the reference frame interval, a reference frame interval of 2 seconds or less produces the best results. The default reference frame interval is every 15th frame (once a second at the default frame rate of 15 frames per second).

MCI_DGV_SET_BRIGHTNESS
Sets the brightness level in the range 0-100.

MCI_DGV_SET_CONTRAST
Sets the contrast level in the range 0-100.

MCI_DGV_SET_HUE
Sets the hue level in the range 0-100, where 0 indicates maximum green tint, 100 indicates maximum red tint, and 50 indicates a neutral tint.

MCI_DGV_SET_SATURATION
Sets the saturation level in the range 0-100.

MCI_DGV_SET_VIDEO_QUALITY
Specifies the compression quality level setting to be sent to the CODEC. This value is in the range 0-10,000. Not all CODECs support setting a quality level. The default setting for video quality is 5000.

MCI_DGV_SET_MONITOR
Sets monitoring of the incoming video signal on or off. Must be used in conjunction with MCI_SET_ON or MCI_SET_OFF. The default setting is MCI_OFF.

When monitoring is turned on, a monitor window is created. Monitor window function is similar to that of the playback window: half, normal, double size, clipping, and so on. When the monitor window is active and recording is not in progress, the monitor window will display the entire video source image, regardless of any source rectangle setting. During recording, only the area being captured is displayed.

If a recording source rectangle is set, the monitor window continues to display the entire video source image with the source capture rectangle displayed in the monitor window image as an animated dashed-line rectangle (unless the source rectangle is the entire video source extent, that is, the entire image is to be captured, in which case the dashed-line rectangle is not displayed). The recording source rectangle may be set directly on the monitor window image by pointing to one corner of the area to be captured, pressing and holding the left mouse button to expand the rectangle to the opposite corner, and then releasing the left mouse button. The dashed-line rectangle will track the mouse movement while the button is held, and will "snap" to the nearest allowable rectangle size.

Monitoring during real-time recording is supported but at a reduced performance. Monitoring can not be turned on or off during recording, that is, if it is on when recording starts it must remain on while recording is in progress; if it is off it must remain off while recording is in progress. Attempting to turn monitoring on or off during real-time recording will result in an MCIERR_INVALID_MODE return. Monitoring during frame-step recording is an application function.

During monitoring, audio is passed through and heard on the speakers or headphones connected to the sound card, if present.

MCI_DGV_SET_CHANNELS
Sets the number of channels in the audio soundtrack recording (1 = mono, 2 = stereo). The default setting is 1.

MCI_DGV_SET_SAMPLESPERSEC
Sets the number of waveform samples per second in the audio soundtrack recording. This value is usually 11025, 22050, or 44100. The default is 11025.

MCI_DGV_SET_BITSPERSAMPLE
Sets the waveform sample size for the audio soundtrack recording. This value is usually 8 or 16 (bits). The default is 8.

MCI_DGV_SET_TRANSPARENT_COLOR
Sets the transparent color used as the *chroma-key* value for transparency in graphics on video overlay hardware devices. Specifying this item has the same effect as specifying MCI_DGV_SET_GRAPHIC_TRANSPARENT_COLOR. Video will be seen wherever the transparency color is painted in graphics. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

MCI_DGV_SET_GRAPHIC_TRANSPARENT_COLOR
Sets the transparent color (used as the *chroma-key* value) for transparency in graphics on video-overlay hardware devices. Specifying this item has the same effect as specifying MCI_DGV_SET_TRANSPARENT_COLOR. Video will be seen wherever the transparency color is painted in graphics. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

MCI_DGV_SET_VIDEO_TRANSPARENT_COLOR
Sets transparency color for transparency in video on dual-plane hardware devices. Graphics will be seen wherever the transparency color appears in the video. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

Note: Transparency color settings apply to both monitor and playback windows for a device instance, and while transparency values are maintained on a per-instance basis, most dual-plane video adapters only allow for a single setting that is applied to the entire screen. Default values for transparency colors are stored in a device .INI file.

MCI_DGV_SET_VIDEO_RECORD_RATE
: Sets the frame rate for recording as an integral number of frames per second in the range 0-30. This sets the target capture rate, but there is no guarantee this rate will be attained. Drop frame records will be inserted into the output data stream to indicate frames dropped during the record process. The default record frame rate is 15 frames per second.

MCI_DGV_SET_VIDEO_RECORD_FRAME_DURATION
: Sets the frame rate for recording as the time duration of each frame in microseconds. This is useful for setting non-integer frame rates, for example, 12.5 frames per second of a PAL videodisc: 1000000/12.5 = 8000 microseconds. The default frame duration is 66,667 microseconds (equivalent to 15 frames per second).

The following additional time formats are supported by digital video devices and can be specified as values for the *ulTimeFormat* of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_MILLISECONDS
: Changes the time format to milliseconds.

MCI_FORMAT_MMTIME
: Changes the time format to MMTIME.

MCI_FORMAT_FRAMES
: Changes the time format to frames.

MCI_FORMAT_HMS
: Changes the time format to hours, minutes, seconds.

MCI_FORMAT_HMSF
: Changes the time format to hours, minutes, seconds, and frames.

Sequencer Extensions

The following additional flags apply to MIDI sequencer devices. All sequencer flags are mutually exclusive, because only one set function can be performed at a time with the MCI_SET message.

MCI_SEQ_SET_MASTER
: Sets the sequencer as a source of synchronization data and indicates that the type of synchronization is specified in the *ulMaster* field of the data structure identified by *pParam2*. The following constants are defined for the synchronization type:

    MCI_SEQ_MIDI
    : The sequencer will send MIDI format synchronization data.

    MCI_SEQ_SMPTE
    : The sequencer will send SMPTE format synchronization data.

    MCI_SEQ_NONE
    : The sequencer will not send synchronization data.

MCI_SEQ_SET_OFFSET
: Changes the SMPTE offset of a sequencer to that specified by the *ulOffset* field of the structure pointed to by *pParam2*. This only affects sequences with a SMPTE division type.

MCI_SEQ_SET_PORT
: Sets the output MIDI port of a sequencer to that specified by the MIDI device ID in the *ulPort* field of the data structure identified by *pParam2*. The device will close the previous port (if any), and attempt to open and use the new port. If it fails, it will return an error and reopen the previously used port (if any). The following constants are defined for the ports:

    MCI_SET_NONE
    : Closes the previously used port (if any). The sequencer will behave exactly the same as if a port were open, except no MIDI message will be sent.

    MIDI_MAPPER
    : Sets the port opened to the MIDI Mapper.

MCI_SEQ_SET_SLAVE
: Sets the sequencer to receive synchronization data and indicates the type of synchronization is specified in the *ulSlave* field of the data structure pointed to by *pParam2*. The following constants are defined for the synchronization type:

MCI_SEQ_FILE
Sets the sequencer to receive synchronization data contained in the MIDI file.

MCI_SEQ_MIDI
Sets the sequencer to receive MIDI format synchronization data.

MCI_SEQ_SMPTE
Sets the sequencer to receive SMPTE format synchronization data.

MCI_SEQ_NONE
Sets the sequencer to ignore synchronization data in a MIDI stream.

MCI_SEQ_SET_TEMPO
Changes the tempo of the MIDI sequence to that specified by the *ulTempo* field of the structure pointed to by *pParam2*. For sequences with division type PPQN, tempo is specified in beats per minute; for sequences with division type SMPTE, tempo is specified in frames per second. This function is not currently supported by the IBM sequencer.

The following additional time-format flags apply to MIDI devices:

MCI_SEQ_SET_SMPTE_24
Sets the time format to 24 frame SMPTE.

MCI_SEQ_SET_SMPTE_25
Sets the time format to 25 frame SMPTE.

MCI_SEQ_SET_SMPTE_30
Sets the time format to 30 frame SMPTE.

MCI_SEQ_SET_SMPTE_30DROP
Sets the time format to 30 drop-frame SMPTE.

MCI_SEQ_SET_SONGPTR
Sets the time format to song pointer units.

Videodisc Extensions

The following additional flags apply to videodisc devices:

MCI_VD_SET_CHANNEL
This flag sets the video channel to the channel specified in *ulChannel* of MCI_VD_SET_PARMS.

MCI_VD_SET_VIDEO
This flag sets Video.

MCI_VD_SET_DISPLAY
This flag sets the display index.

MCI_VD_SET_ON
This flag sets videodisc driver ON.

MCI_VD_SET_OFF
This flag sets videodisc driver OFF.

The following additional time formats apply to videodisc devices and can be specified as values for the *ulTimeFormat* field of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_CHAPTERS
This flag changes the time format to chapters.

MCI_FORMAT_FRAMES
This flag changes the time format to frames.

MCI_FORMAT_HMS
This flag changes the time format to hours, minutes, and seconds.

MCI_FORMAT_HMSF
This flag changes the time format to hours, minutes, seconds, and frames.

The MCI_VD_SET_CHANNEL and MCI_VD_SET_VIDEO flags are mutually exclusive and must be used with the MCI_VD_SET_ON and MCI_VD_SET_OFF flags.

Video Overlay Extensions

The following additional items apply to video overlay devices and can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_OVLY_SET_IMAGE_FILE_FORMAT
: Sets the specified image file format in which the image capture is to be stored (when saved). This format must be specified by a four-character code (for example, MMOT or OS13), and must be one of the currently supported and installed MMIO image file formats, or the device-specific format. This does not effect the loading or restoring of images. It overwrites any previous file-format value, such as that obtained through a LOAD operation.

MCI_OVLY_SET_IMAGE_COMPRESSION
: This flag sets the compression type used for saving still images. The specified compression type is used if it is supported by the device, the file format, or both. The compression type is not used if it contradicts settings for file format, BITSPERPEL, or PELFORMAT.

  If the compression type value is valid, it supersedes any image quality setting and overwrites any format tag or compression value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

  Compression algorithms are often proprietary and can require hardware assistance for performance. Therefore, when possible, the setting of this item is controlled by the device. If the specified compression type is not compatible with file format or BITSPERPEL settings, the device selects a compression type based on the file format, PELFORMAT, and quality settings.

  If the compression type is not available, the device returns "function not supported" and uses the current setting.

  M-Motion specific: The default is MCI_IMG_COMP_NONE.

MCI_OVLY_SET_IMAGE_BITSPERPEL
: Sets the number of bits per pixel used for the image file to be saved. Generally supported values are those defined for OS/2 2.0 bit maps. Some devices might support other values.

  The value specified for this setting might not be the same as the number of colors currently visible on the display. Selecting a BITSPERPEL value greater than that currently displayed results in unused colors. Selecting a BITSPERPEL value less than that currently displayed results in a degradation of color and a reduced quality image.

  Most file formats do not support all BITSPERPEL values. This item overwrites any BITSPERPEL value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

  Some devices are not capable of adjusting the number of colors to be saved in the image file. When this is the case, the device captures and saves the image in whatever number of colors it supports. The actual value used can be obtained using the MCI_OVLY_STATUS_IMAGE_BITSPERPEL flag.

  If variable BITSPERPEL representation is not available, the device returns "function not supported" and uses the current setting.

  M-Motion specific: The default is 12.

MCI_OVLY_SET_IMAGE_PELFORMAT
: This flag sets the pixel format used for saving bit maps. This value indicates the desired image file color representation, and is used in conjunction with the BITSPERPEL value. Supported pixel format values are:

  MCI_IMG_PALETTE
  : A palettized video image with 1, 4, or 8 bits per pixel.

  MCI_IMG_RGB
  : An RGB video image with 16 or 24 bits per pixel.

  MCI_IMG_YUV
  : A YUVB video image with 9, 12, or 16 bits per pixel.

  Most file formats do not support all pixel formats. This item overwrites any pixel format value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

  Some devices are not capable of adjusting the color representation of the image. When this is the case, the device captures and saves the image in whatever color representation it supports. If variable color representation is not available, the device returns "function not supported" and uses the current setting.

  M-Motion specific: The default is MCI_IMG_YUV.

MCI_OVLY_SET_BRIGHTNESS
: This flag sets the brightness level in the range 0-100.

MCI_OVLY_SET_CONTRAST
This flag sets the contrast level in the range 0-100.

MCI_OVLY_SET_HUE
This flag sets the hue level in the range 0-100. A value of 50 indicates neutral tint.

MCI_OVLY_SET_SATURATION
This flag sets the saturation level in the range 0-100.

MCI_OVLY_SET_SHARPNESS
This flag sets the sharpness level in the range 0-100.

MCI_OVLY_SET_GREYSCALE
This flag turns the grey scale on or off. Must be used in conjunction with MCI_SET_ON or MCI_SET_OFF.

MCI_OVLY_SET_IMAGE_QUALITY
This flag sets the specified image quality level. This item indicates the perceived quality of the image to be saved and allows the device to select the most appropriate compression method when more than one is available. The value specified for this item can affect the size of the resulting file.

This item overwrites any quality value obtained by a LOAD operation. However, it does not affect the loading or restoring of images. If image quality is not previously set, the device selects a compression scheme as accurately as possible.

If variable image quality is not available, the device returns "function not supported" and uses the current setting.

Supported values are:

MCI_IMG_QUALITY_HIGH
This flag normally describes photo-realistic images with high resolution and color content.

MCI_IMG_QUALITY_MED
This flag normally describes images such as complete graphs, charts, or diagrams, with fewer color transitions and complexity.

MCI_IMG_QUALITY_LOW
This flag normally describes images such as cartoons and simple drawings.

M-Motion specific: The default is MCI_IMG_QUALITY_HIGH.

MCI_OVLY_SET_IMAGE_COMPRESSION_METHOD
This flag sets the method by which image compression or decompression is done. Supported values and their meanings are:

MCI_CODEC_DEFAULT
This flag selects the default compression method specified in the INI file.

MCI_CODEC_SW_ONLY
This flag selects to use software emulation as the compression method.

MCI_CODEC_HW
This flag selects to use the compression method supported by the hardware, if available. Otherwise, software emulation is used.

MCI_OVLY_SET_MINIMUM_VIDEO_REFRESH_RATE
This flag sets the minimum refresh rate for the device instance. This is the minimum frame display refresh rate the application will accept for this device instance. This parameter is used on hardware that can *multiplex* the digitization between different windows at reduced rates. The default is one, allowing degraded display on hardware that supports this capability.

Waveform Audio Extensions

The following additional flags apply to wave audio devices and are mutually exclusive. If MCI_WAVE_SET_FORMATTAG is specified, the driver can change other settings to maintain compatibility. After setting the waveform format, the other parameters can be set as necessary within the currently selected waveform format. An error will be returned if the requested change results in an unsupported configuration.

An application can use the MCI_STATUS message to see if any of the other settings were changed to maintain a valid configuration.

MCI_WAVE_SET_FORMATTAG
Sets the format type used for playing, recording, and saving to the *usFormatTag* field of the MCI_WAVE_SET_PARMS data structure. Refer to the RIFF WAVE format documentation for more information.

The following constants are defined to set the format type. Additional subtype values can be found in OS2MEDEF.H.

MCI_WAVE_FORMAT_PCM
: Changes the format to pulse code modulation (PCM).

MCI_WAVE_FORMAT_ADPCM
: Changes the format to adaptive differential pulse code modulation (ADPCM).

MCI_WAVE_FORMAT_IBM_CVSD
: Changes the format to IBM Speech Viewer.

MCI_WAVE_FORMAT_ALAW
: Changes the format to A-Law.

MCI_WAVE_FORMAT_MULAW
: Changes the format to Mu-Law.

MCI_WAVE_FORMAT_IBM_ALAW
: Changes the format to A-Law.

MCI_WAVE_FORMAT_IBM_MULAW
: Changes the format to Mu-Law.

MCI_WAVE_FORMAT_OKI_ADPCM
: Changes the format to OKI ADPCM.

MCI_WAVE_FORMAT_DVI_ADPCM
: Changes the format to DVI ADPCM.

MCI_WAVE_FORMAT_IBM_ADPCM
: Changes the format to ADPCM.

MCI_WAVE_FORMAT_DIGISTD
: Changes the format to IBM Digispeech (standard format).

MCI_WAVE_FORMAT_DIGIFIX
: Changes the format to IBM Digispeech (fixed format).

MCI_WAVE_FORMAT_AVC_ADPCM
: Changes the format to AVC ADPCM.

MCI_WAVE_FORMAT_CT_ADPCM
: Changes the format to Creative Labs ADPCM.

MCI_WAVE_FORMAT_MPEG1
: Changes the format to MPEG audio.

MCI_WAVE_SET_CHANNELS
: Sets the channel count used for playing, recording, and saving to the *usChannels* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_SAMPLESPERSEC
: Sets the samples per second used for playing, recording, and saving to the *ulSamplesPerSec* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_AVGBYTESPERSEC
: Sets the bytes per second used for playing, recording, and saving to the *ulAvgBytesPerSec* field of the MCI_WAVE_SET_PARMS data structure. Playback software may use this number to estimate required buffer sizes.

MCI_WAVE_SET_BLOCKALIGN
: Sets the block alignment used for playing, recording, and saving to the *usBlockAlign* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_BITSPERSAMPLE
: Sets the bits per sample used for playing, recording, and saving to the *usBitsPerSample* field of the MCI_WAVE_SET_PARMS data structure.

The following additional time format flags apply to wave audio devices and can be specified for the *ulTimeFormat* field: for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_SAMPLES

Change time format to samples.

MCI_FORMAT_BYTES
: Change time format to bytes.

MCI_SET Parameter - pParam2

pParam2 (PMCI_SET_PARMS)
A pointer to an MCI_SET_PARMS data structure. (This is the default parameter data structure.) Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_SET_PARMS
: A pointer to the MCI_AMP_SET_PARMS data structure.

PMCI_CDXA_SET_PARMS
: A pointer to the MCI_CDXA_SET_PARMS data structure.

PMCI_DGV_SET_PARMS
: A pointer to the MCI_DGV_SET_PARMS data structure.

PMCI_SEQ_SET_PARMS
: A pointer to the MCI_SEQ_SET_PARMS data structure.

PMCI_VD_SET_PARMS
: A pointer to the MCI_VD_SET_PARMS data structure.

PMCI_OVLY_SET_PARMS
: A pointer to the MCI_OVLY_SET_PARMS data structure.

PMCI_WAVE_SET_PARMS
: A pointer to the MCI_WAVE_SET_PARMS data structure. This data structure replaces the standard default data structure, MCI_SET_PARMS.

MCI_SET Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM/2 driver for this command.

MCIERR_MISSING_FLAG
: Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
: The flags cannot be used together.

MCIERR_MISSING_STRING_ARGUMENT
: Missing required string argument.

MCIERR_INVALID_ITEM_FLAG
: Invalid item flag specified for this command.

MCIERR_INSTANCE_INACTIVE
: Instance inactive.

MCIERR_OUTOFRANGE
: Value given is out of range.

MCIERR_UNSUPPORTED_FUNCTION
: Function not supported.

MCI_SET - Description

This message is used to set device parameters or information.

ulParam1 (ULONG)
This parameter can contain the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_AUDIO
: Sets audio attributes of the device instance. A device with audio capabilities might support both left and right channels. The channel is specified in the *ulAudio* field of the data structure specified by *pParam2*. The action to be taken is specified with the flags MCI_SET_ON (which enables audio output at the current volume level), MCI_SET_OFF (which mutes audio output), or MCI_SET_VOLUME. Specifying MCI_SET_VOLUME does not enable audio output if MCI_SET_OFF has been previously specified.

  The following constants are defined for specifying the audio channel in the *ulAudio* field.

  MCI_SET_AUDIO_ALL
  : Apply to both channels.

  MCI_SET_AUDIO_LEFT
  : Apply to the left channel only.

  MCI_SET_AUDIO_RIGHT
  : Apply to the right channel only.

MCI_SET_DOOR_OPEN
: Instructs the device to open the media cover (if any). This message ejects the media from devices where appropriate.

MCI_SET_DOOR_CLOSED
: Instructs the device to close the media cover (if any).

MCI_SET_DOOR_LOCK
: Locks the media cover on the device (if any). This disables manual ejection of the media from the device.

MCI_SET_DOOR_UNLOCK
: Unlocks the media cover on the device (if any). This enables manual ejection of the media from the device.

MCI_SET_VOLUME
: Sets the level of audio as a percentage of the maximum audio level as indicated in the *ulLevel* field. The volume level that can be set on the device might be of coarser granularity than that specified. In this case, the actual level can be obtained by issuing a MCI_STATUS message. If a number greater than 100 is given, then 100 will be used as the volume setting, and no error will be returned. See Examples section for an example using this flag.

MCI_SET_VIDEO
: Sets the video signal on or off. This flag must be used with either MCI_SET_ON or MCI_SET_OFF.

MCI_SET_ON
: Sets the video or specified audio channel on.

MCI_SET_OFF
: Sets the video or specified audio channel off.

MCI_SET_SPEED_FORMAT
: Specifies the speed format to be used on subsequent commands contained in the *ulSpeedFormat* field. The following values can be used:

    MCI_FORMAT_PERCENTAGE
    : Specifies the subsequent speed values as a percentage of the normal speed.

    MCI_FORMAT_FPS
    : Specifies the subsequent speed values in frames per second. This is the default setting.

MCI_SET_TIME_FORMAT
: Uses a time format on subsequent commands. A time-format parameter must be indicated in the *ulTimeFormat* field of the data structure specified by *pParam2* if this flag is used. The default is MCI_FORMAT_MMTIME. The following time formats are generic; devices can also provide device-specific time units:

    MCI_FORMAT_MILLISECONDS
    : Indicates that all subsequent commands that specify time will do so in milliseconds for both input and output.

    MCI_FORMAT_MMTIME
    : Indicates that all subsequent commands that specify time will do so in MMTIME units for both input and output. This does not apply to command parameters that explicitly specify time units, such as milliseconds on *ulOver*.

MCI_OVER
: Sets the vectored delay time to change the volume (or other attribute) in milliseconds.

MCI_SET_ITEM
: Indicates that the item to be set is specified in the *ulItem* field of the data structure identified by *pParam2*. Any value associated with the item is contained in the *ulValue* field. Each item defines the use (if any) and meaning of the value in the *ulValue* field.

Amplifier Mixer Extensions

The following additional flags apply to amplifier-mixer devices. Only one audio attribute set function can be performed at a time with the MCI_SET message. The treble, bass, balance, pitch, and gain flags require the MCI_SET_AUDIO flag also to be set. The level to be set for each function is contained in the *ulLevel* field and represents a percentage of the maximum available audio effect provided by the device. Zero is the minimum effect, while 100 is the maximum effect.

The following audio effects apply to the final output mix. Any specification of a particular channel will be ignored.

MCI_AMP_SET_BALANCE
: Sets the final output balance. Zero is defined as full left balance while one hundred is defined as full right balance.

MCI_AMP_SET_BASS
: Controls bass as a percentage of the maximum achievable effect.

MCI_AMP_SET_GAIN
: Sets the gain as a percentage of the maximum achievable effect.

MCI_AMP_SET_PITCH
: Sets the pitch as a percentage of the maximum achievable effect.

MCI_AMP_SET_TREBLE
Controls treble as a percentage of the maximum achievable effect.

The following items can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_AMP_SET_AUDIO
Used with the extended ampmix audio attribute flags.

MCI_AMP_SET_MONITOR
Used with the MCI_SET_ON or MCI_SET_OFF flags. It instructs the ampmix device to monitor the currently selected connector. This flag is typically used to listen to (monitor) a source while it is being recorded by another device.

If the MCI_SET_ITEM flag is set and MCI_AMP_SET_AUDIO is in the *ulItem* field of MCI_AMP_SET_PARMS, the connector specified in *ulValue* can be modified with the following audio attribute flags in *ulAudio* and the appropriate level in *ulLevel*.

MCI_AMP_SET_ALC
The *ulLevel* field in MCI_AMP_SET_PARMS contains the auto-level control setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_BALANCE
The *ulLevel* field in MCI_AMP_SET_PARMS contains the balance setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_BASS
The *ulLevel* field in MCI_AMP_SET_PARMS contains the bass setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CHORUS
The *ulLevel* field in MCI_AMP_SET_PARMS contains the chorus setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CROSSOVER
The *ulLevel* field in MCI_AMP_SET_PARMS contains the crossover setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM1
The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100). for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM2
The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_CUSTOM3
The *ulLevel* field in MCI_AMP_SET_PARMS contains the custom effect setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_GAIN
The *ulLevel* field in MCI_AMP_SET_PARMS contains the gain setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_LOUDNESS
The *ulLevel* field in MCI_AMP_SET_PARMS contains the loudness setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MID
The *ulLevel* field in MCI_AMP_SET_PARMS contains the mid setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MONITOR
The *ulLevel* field in MCI_AMP_SET_PARMS contains the monitor setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_MUTE
The *ulLevel* field in MCI_AMP_SET_PARMS contains the mute setting for the connector specified in *ulValue*.

MCI_AMP_SET_PITCH
The *ulLevel* field in MCI_AMP_SET_PARMS contains the pitch setting as a percentage (0-100) for the connector specified in *ulValue*.

MCI_AMP_SET_REVERB
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the reverb setting as a percentage (0-100) for the connector specified in *ulValue* .

MCI_AMP_SET_STEREOENHANCE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the stereo enhance setting as a percentage (0-100) for the connector specified in *ulValue* .

MCI_AMP_SET_TREBLE
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the treble setting as a percentage (0-100) for the connector specified in *ulValue* .

MCI_AMP_SET_VOLUME
: The *ulLevel* field in MCI_AMP_SET_PARMS contains the volume setting as a percentage (0-100) for the connector specified in *ulValue* .

CD Audio Extensions

The following additional time formats are supported by CD audio devices and can be specified as values for the *ulTimeFormat* field of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_MSF
: Indicates that all subsequent commands that specify time will do so in *mm:ss:ff* where *mm* is minutes, *ss* is seconds and *ff* is frames.

MCI_FORMAT_TMSF
: Indicates that all subsequent commands that specify time will do so in *tt:mm:ss:ff* where *tt* is tracks, *mm* is minutes, *ss* is seconds, and ff is frames.

CD/XA Extensions

The following additional flags apply to the CD/XA device. Only one channel set function can be performed at a time with the MCI_SET message. The channel is specified in the *ulChannel* field of the data structure. The destination of the data in that channel is determined by the flags below. Only one destination can be selected at a time with the MCI_SET message. This message must be used with the MCI_CDXA_SET_CHANNEL flag and either the MCI_SET_ON or MCI_SET_OFF flags.

MCI_CDXA_AUDIO_DEVICE
: Sends the audio stream to the audio card.

MCI_CDXA_AUDIO_BUFFER
: Sends the audio stream to a playlist.

MCI_CDXA_VIDEO_BUFFER
: Sends the video stream to a playlist.

MCI_CDXA_DATA_BUFFER
: Sends the data stream to a playlist.

Digital Video Extensions

The following additional items can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_DGV_SET_VIDEO_COMPRESSION
: Specifies the FOURCC compression format used for recording digital motion video. The values that can be specified are:

  MCI_VID_COMP_ULTI
  : Ultimotion

  MCI_VID_COMP_DIB
  : Raw (uncompressed format)

  MCI_VID_COMP_RT21
  : Indeo 2.1

  MCI_VID_COMP_IV31
  : Indeo 3.1

  The default compression type is specified through the Setup page for the digital video device. The initial setting is MCI_VID_COMP_ULTI until changed in the Setup.

  Note: Compressors are not available for FLIC, MPEG, and Indeo 3.2 in this version of OS/2.

MCI_DGV_SET_RECORD_AUDIO
Sets audio soundtrack recording on or off. The default is MCI_ON. This flag is used with MCI_ON or MCI_OFF.

MCI_DGV_SET_REF_INTERVAL
Sets the frequency at which reference frames (or I-frames) are to be compressed in the output data stream. A value of 0 results in no I-frames, a value of 1 causes every frame to be an I-frame, a value of 2 causes every other frame to be an I-frame, and so on. While there is no upper bound on the reference frame interval, a reference frame interval of 2 seconds or less produces the best results. The default reference frame interval is every 15th frame (once a second at the default frame rate of 15 frames per second).

MCI_DGV_SET_BRIGHTNESS
Sets the brightness level in the range 0-100.

MCI_DGV_SET_CONTRAST
Sets the contrast level in the range 0-100.

MCI_DGV_SET_HUE
Sets the hue level in the range 0-100, where 0 indicates maximum green tint, 100 indicates maximum red tint, and 50 indicates a neutral tint.

MCI_DGV_SET_SATURATION
Sets the saturation level in the range 0-100.

MCI_DGV_SET_VIDEO_QUALITY
Specifies the compression quality level setting to be sent to the CODEC. This value is in the range 0-10,000. Not all CODECs support setting a quality level. The default setting for video quality is 5000.

MCI_DGV_SET_MONITOR
Sets monitoring of the incoming video signal on or off. Must be used in conjunction with MCI_SET_ON or MCI_SET_OFF. The default setting is MCI_OFF.

When monitoring is turned on, a monitor window is created. Monitor window function is similar to that of the playback window: half, normal, double size, clipping, and so on. When the monitor window is active and recording is not in progress, the monitor window will display the entire video source image, regardless of any source rectangle setting. During recording, only the area being captured is displayed.

If a recording source rectangle is set, the monitor window continues to display the entire video source image with the source capture rectangle displayed in the monitor window image as an animated dashed-line rectangle (unless the source rectangle is the entire video source extent, that is, the entire image is to be captured, in which case the dashed-line rectangle is not displayed). The recording source rectangle may be set directly on the monitor window image by pointing to one corner of the area to be captured, pressing and holding the left mouse button to expand the rectangle to the opposite corner, and then releasing the left mouse button. The dashed-line rectangle will track the mouse movement while the button is held, and will "snap" to the nearest allowable rectangle size.

Monitoring during real-time recording is supported but at a reduced performance. Monitoring can not be turned on or off during recording, that is, if it is on when recording starts it must remain on while recording is in progress; if it is off it must remain off while recording is in progress. Attempting to turn monitoring on or off during real-time recording will result in an MCIERR_INVALID_MODE return. Monitoring during frame-step recording is an application function.

During monitoring, audio is passed through and heard on the speakers or headphones connected to the sound card, if present.

MCI_DGV_SET_CHANNELS
Sets the number of channels in the audio soundtrack recording (1 = mono, 2 = stereo). The default setting is 1.

MCI_DGV_SET_SAMPLESPERSEC
Sets the number of waveform samples per second in the audio soundtrack recording. This value is usually 11025, 22050, or 44100. The default is 11025.

MCI_DGV_SET_BITSPERSAMPLE
Sets the waveform sample size for the audio soundtrack recording. This value is usually 8 or 16 (bits). The default is 8.

MCI_DGV_SET_TRANSPARENT_COLOR
Sets the transparent color used as the *chroma-key* value for transparency in graphics on video overlay hardware devices. Specifying this item has the same effect as specifying MCI_DGV_SET_GRAPHIC_TRANSPARENT_COLOR. Video will be seen wherever the transparency color is painted in graphics. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

MCI_DGV_SET_GRAPHIC_TRANSPARENT_COLOR

Sets the transparent color (used as the *chroma-key* value) for transparency in graphics on video-overlay hardware devices. Specifying this item has the same effect as specifying MCI_DGV_SET_TRANSPARENT_COLOR. Video will be seen wherever the transparency color is painted in graphics. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

MCI_DGV_SET_VIDEO_TRANSPARENT_COLOR
Sets transparency color for transparency in video on dual-plane hardware devices. Graphics will be seen wherever the transparency color appears in the video. The color is set as a numeric value in the range 0...(*n* - 1). Where *n* represents the number of available colors.

Note: Transparency color settings apply to both monitor and playback windows for a device instance, and while transparency values are maintained on a per-instance basis, most dual-plane video adapters only allow for a single setting that is applied to the entire screen. Default values for transparency colors are stored in a device .INI file.

MCI_DGV_SET_VIDEO_RECORD_RATE
Sets the frame rate for recording as an integral number of frames per second in the range 0-30. This sets the target capture rate, but there is no guarantee this rate will be attained. Drop frame records will be inserted into the output data stream to indicate frames dropped during the record process. The default record frame rate is 15 frames per second.

MCI_DGV_SET_VIDEO_RECORD_FRAME_DURATION
Sets the frame rate for recording as the time duration of each frame in microseconds. This is useful for setting non-integer frame rates, for example, 12.5 frames per second of a PAL videodisc: 1000000/12.5 = 8000 microseconds. The default frame duration is 66,667 microseconds (equivalent to 15 frames per second).

The following additional time formats are supported by digital video devices and can be specified as values for the *ulTimeFormat* of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_MILLISECONDS
Changes the time format to milliseconds.

MCI_FORMAT_MMTIME
Changes the time format to MMTIME.

MCI_FORMAT_FRAMES
Changes the time format to frames.

MCI_FORMAT_HMS
Changes the time format to hours, minutes, seconds.

MCI_FORMAT_HMSF
Changes the time format to hours, minutes, seconds, and frames.

Sequencer Extensions

The following additional flags apply to MIDI sequencer devices. All sequencer flags are mutually exclusive, because only one set function can be performed at a time with the MCI_SET message.

MCI_SEQ_SET_MASTER
Sets the sequencer as a source of synchronization data and indicates that the type of synchronization is specified in the *ulMaster* field of the data structure identified by *pParam2*. The following constants are defined for the synchronization type:

MCI_SEQ_MIDI
The sequencer will send MIDI format synchronization data.

MCI_SEQ_SMPTE
The sequencer will send SMPTE format synchronization data.

MCI_SEQ_NONE
The sequencer will not send synchronization data.

MCI_SEQ_SET_OFFSET
Changes the SMPTE offset of a sequencer to that specified by the *ulOffset* field of the structure pointed to by *pParam2*. This only affects sequences with a SMPTE division type.

MCI_SEQ_SET_PORT
Sets the output MIDI port of a sequencer to that specified by the MIDI device ID in the *ulPort* field of the data structure identified by *pParam2*. The device will close the previous port (if any), and attempt to open and use the new port. If it fails, it will return an error and reopen the previously used port (if any). The following constants are defined for the ports:

MCI_SET_NONE
Closes the previously used port (if any). The sequencer will behave exactly the same as if a port were open, except no MIDI message will be sent.

MIDI_MAPPER
Sets the port opened to the MIDI Mapper.

MCI_SEQ_SET_SLAVE
Sets the sequencer to receive synchronization data and indicates the type of synchronization is specified in the *ulSlave* field of the data structure pointed to by *pParam2*. The following constants are defined for the synchronization type:

MCI_SEQ_FILE
Sets the sequencer to receive synchronization data contained in the MIDI file.

MCI_SEQ_MIDI
Sets the sequencer to receive MIDI format synchronization data.

MCI_SEQ_SMPTE
Sets the sequencer to receive SMPTE format synchronization data.

MCI_SEQ_NONE
Sets the sequencer to ignore synchronization data in a MIDI stream.

MCI_SEQ_SET_TEMPO
Changes the tempo of the MIDI sequence to that specified by the *ulTempo* field of the structure pointed to by *pParam2*. For sequences with division type PPQN, tempo is specified in beats per minute; for sequences with division type SMPTE, tempo is specified in frames per second. This function is not currently supported by the IBM sequencer.

The following additional time-format flags apply to MIDI devices:

MCI_SEQ_SET_SMPTE_24
Sets the time format to 24 frame SMPTE.

MCI_SEQ_SET_SMPTE_25
Sets the time format to 25 frame SMPTE.

MCI_SEQ_SET_SMPTE_30
Sets the time format to 30 frame SMPTE.

MCI_SEQ_SET_SMPTE_30DROP
Sets the time format to 30 drop-frame SMPTE.

MCI_SEQ_SET_SONGPTR
Sets the time format to song pointer units.

Videodisc Extensions

The following additional flags apply to videodisc devices:

MCI_VD_SET_CHANNEL
This flag sets the video channel to the channel specified in *ulChannel* of MCI_VD_SET_PARMS.

MCI_VD_SET_VIDEO
This flag sets Video.

MCI_VD_SET_DISPLAY
This flag sets the display index.

MCI_VD_SET_ON
This flag sets videodisc driver ON.

MCI_VD_SET_OFF
This flag sets videodisc driver OFF.

The following additional time formats apply to videodisc devices and can be specified as values for the *ulTimeFormat* field of the data structure pointed to by *pParam2* for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_CHAPTERS
This flag changes the time format to chapters.

MCI_FORMAT_FRAMES

This flag changes the time format to frames.

MCI_FORMAT_HMS
This flag changes the time format to hours, minutes, and seconds.

MCI_FORMAT_HMSF
This flag changes the time format to hours, minutes, seconds, and frames.

The MCI_VD_SET_CHANNEL and MCI_VD_SET_VIDEO flags are mutually exclusive and must be used with the MCI_VD_SET_ON and MCI_VD_SET_OFF flags.

Video Overlay Extensions

The following additional items apply to video overlay devices and can be specified for the *ulItem* field of the data structure pointed to by *pParam2* for use with the MCI_SET_ITEM flag:

MCI_OVLY_SET_IMAGE_FILE_FORMAT
Sets the specified image file format in which the image capture is to be stored (when saved). This format must be specified by a four-character code (for example, MMOT or OS13), and must be one of the currently supported and installed MMIO image file formats, or the device-specific format. This does not effect the loading or restoring of images. It overwrites any previous file-format value, such as that obtained through a LOAD operation.

MCI_OVLY_SET_IMAGE_COMPRESSION
This flag sets the compression type used for saving still images. The specified compression type is used if it is supported by the device, the file format, or both. The compression type is not used if it contradicts settings for file format, BITSPERPEL, or PELFORMAT.

If the compression type value is valid, it supersedes any image quality setting and overwrites any format tag or compression value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

Compression algorithms are often proprietary and can require hardware assistance for performance. Therefore, when possible, the setting of this item is controlled by the device. If the specified compression type is not compatible with file format or BITSPERPEL settings, the device selects a compression type based on the file format, PELFORMAT, and quality settings.

If the compression type is not available, the device returns "function not supported" and uses the current setting.

M-Motion specific: The default is MCI_IMG_COMP_NONE.

MCI_OVLY_SET_IMAGE_BITSPERPEL
Sets the number of bits per pixel used for the image file to be saved. Generally supported values are those defined for OS/2 2.0 bit maps. Some devices might support other values.

The value specified for this setting might not be the same as the number of colors currently visible on the display. Selecting a BITSPERPEL value greater than that currently displayed results in unused colors. Selecting a BITSPERPEL value less than that currently displayed results in a degradation of color and a reduced quality image.

Most file formats do not support all BITSPERPEL values. This item overwrites any BITSPERPEL value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

Some devices are not capable of adjusting the number of colors to be saved in the image file. When this is the case, the device captures and saves the image in whatever number of colors it supports. The actual value used can be obtained using the MCI_OVLY_STATUS_IMAGE_BITSPERPEL flag.

If variable BITSPERPEL representation is not available, the device returns "function not supported" and uses the current setting.

M-Motion specific: The default is 12.

MCI_OVLY_SET_IMAGE_PELFORMAT
This flag sets the pixel format used for saving bit maps. This value indicates the desired image file color representation, and is used in conjunction with the BITSPERPEL value. Supported pixel format values are:

MCI_IMG_PALETTE
A palettized video image with 1, 4, or 8 bits per pixel.

MCI_IMG_RGB
An RGB video image with 16 or 24 bits per pixel.

MCI_IMG_YUV
A YUVB video image with 9, 12, or 16 bits per pixel.

Most file formats do not support all pixel formats. This item overwrites any pixel format value obtained by a LOAD operation. However, it does not affect the loading or restoring of images.

Some devices are not capable of adjusting the color representation of the image. When this is the case, the device captures and saves the image in whatever color representation it supports. If variable color representation is not available, the device returns "function not supported" and uses the current setting.

M-Motion specific: The default is MCI_IMG_YUV.

MCI_OVLY_SET_BRIGHTNESS
This flag sets the brightness level in the range 0-100.

MCI_OVLY_SET_CONTRAST
This flag sets the contrast level in the range 0-100.

MCI_OVLY_SET_HUE
This flag sets the hue level in the range 0-100. A value of 50 indicates neutral tint.

MCI_OVLY_SET_SATURATION
This flag sets the saturation level in the range 0-100.

MCI_OVLY_SET_SHARPNESS
This flag sets the sharpness level in the range 0-100.

MCI_OVLY_SET_GREYSCALE
This flag turns the grey scale on or off. Must be used in conjunction with MCI_SET_ON or MCI_SET_OFF.

MCI_OVLY_SET_IMAGE_QUALITY
This flag sets the specified image quality level. This item indicates the perceived quality of the image to be saved and allows the device to select the most appropriate compression method when more than one is available. The value specified for this item can affect the size of the resulting file.

This item overwrites any quality value obtained by a LOAD operation. However, it does not affect the loading or restoring of images. If image quality is not previously set, the device selects a compression scheme as accurately as possible.

If variable image quality is not available, the device returns "function not supported" and uses the current setting.

Supported values are:

MCI_IMG_QUALITY_HIGH
This flag normally describes photo-realistic images with high resolution and color content.

MCI_IMG_QUALITY_MED
This flag normally describes images such as complete graphs, charts, or diagrams, with fewer color transitions and complexity.

MCI_IMG_QUALITY_LOW
This flag normally describes images such as cartoons and simple drawings.

M-Motion specific: The default is MCI_IMG_QUALITY_HIGH.

MCI_OVLY_SET_IMAGE_COMPRESSION_METHOD
This flag sets the method by which image compression or decompression is done. Supported values and their meanings are:

MCI_CODEC_DEFAULT
This flag selects the default compression method specified in the INI file.

MCI_CODEC_SW_ONLY
This flag selects to use software emulation as the compression method.

MCI_CODEC_HW
This flag selects to use the compression method supported by the hardware, if available. Otherwise, software emulation is used.

MCI_OVLY_SET_MINIMUM_VIDEO_REFRESH_RATE
This flag sets the minimum refresh rate for the device instance. This is the minimum frame display refresh rate the application will accept for this device instance. This parameter is used on hardware that can *multiplex* the digitization between different windows at reduced rates. The default is one, allowing degraded display on hardware that supports this capability.

Waveform Audio Extensions

The following additional flags apply to wave audio devices and are mutually exclusive. If MCI_WAVE_SET_FORMATTAG is specified, the driver can change other settings to maintain compatibility. After setting the waveform format, the other parameters can be set as necessary within the currently selected waveform format. An error will be returned if the requested change results in an unsupported configuration.

An application can use the MCI_STATUS message to see if any of the other settings were changed to maintain a valid configuration.

MCI_WAVE_SET_FORMATTAG
: Sets the format type used for playing, recording, and saving to the *usFormatTag* field of the MCI_WAVE_SET_PARMS data structure. Refer to the RIFF WAVE format documentation for more information. The following constants are defined to set the format type. Additional subtype values can be found in OS2MEDEF.H.

  MCI_WAVE_FORMAT_PCM
  : Changes the format to pulse code modulation (PCM).

  MCI_WAVE_FORMAT_ADPCM
  : Changes the format to adaptive differential pulse code modulation (ADPCM).

  MCI_WAVE_FORMAT_IBM_CVSD
  : Changes the format to IBM Speech Viewer.

  MCI_WAVE_FORMAT_ALAW
  : Changes the format to A-Law.

  MCI_WAVE_FORMAT_MULAW
  : Changes the format to Mu-Law.

  MCI_WAVE_FORMAT_IBM_ALAW
  : Changes the format to A-Law.

  MCI_WAVE_FORMAT_IBM_MULAW
  : Changes the format to Mu-Law.

  MCI_WAVE_FORMAT_OKI_ADPCM
  : Changes the format to OKI ADPCM.

  MCI_WAVE_FORMAT_DVI_ADPCM
  : Changes the format to DVI ADPCM.

  MCI_WAVE_FORMAT_IBM_ADPCM
  : Changes the format to ADPCM.

  MCI_WAVE_FORMAT_DIGISTD
  : Changes the format to IBM Digispeech (standard format).

  MCI_WAVE_FORMAT_DIGIFIX
  : Changes the format to IBM Digispeech (fixed format).

  MCI_WAVE_FORMAT_AVC_ADPCM
  : Changes the format to AVC ADPCM.

  MCI_WAVE_FORMAT_CT_ADPCM
  : Changes the format to Creative Labs ADPCM.

  MCI_WAVE_FORMAT_MPEG1
  : Changes the format to MPEG audio.

MCI_WAVE_SET_CHANNELS
: Sets the channel count used for playing, recording, and saving to the *usChannels* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_SAMPLESPERSEC
: Sets the samples per second used for playing, recording, and saving to the *ulSamplesPerSec* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_AVGBYTESPERSEC
: Sets the bytes per second used for playing, recording, and saving to the *ulAvgBytesPerSec* field of the MCI_WAVE_SET_PARMS data structure. Playback software may use this number to estimate required buffer sizes.

MCI_WAVE_SET_BLOCKALIGN

Sets the block alignment used for playing, recording, and saving to the *usBlockAlign* field of the MCI_WAVE_SET_PARMS data structure.

MCI_WAVE_SET_BITSPERSAMPLE
: Sets the bits per sample used for playing, recording, and saving to the *usBitsPerSample* field of the MCI_WAVE_SET_PARMS data structure.

The following additional time format flags apply to wave audio devices and can be specified for the *ulTimeFormat* field: for use with the MCI_SET_TIME_FORMAT flag:

MCI_FORMAT_SAMPLES
: Change time format to samples.

MCI_FORMAT_BYTES
: Change time format to bytes.

pParam2 (PMCI_SET_PARMS)

A pointer to an MCI_SET_PARMS data structure. (This is the default parameter data structure.) Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_AMP_SET_PARMS
: A pointer to the MCI_AMP_SET_PARMS data structure.

PMCI_CDXA_SET_PARMS
: A pointer to the MCI_CDXA_SET_PARMS data structure.

PMCI_DGV_SET_PARMS
: A pointer to the MCI_DGV_SET_PARMS data structure.

PMCI_SEQ_SET_PARMS
: A pointer to the MCI_SEQ_SET_PARMS data structure.

PMCI_VD_SET_PARMS
: A pointer to the MCI_VD_SET_PARMS data structure.

PMCI_OVLY_SET_PARMS
: A pointer to the MCI_OVLY_SET_PARMS data structure.

PMCI_WAVE_SET_PARMS
: A pointer to the MCI_WAVE_SET_PARMS data structure. This data structure replaces the standard default data structure, MCI_SET_PARMS.

rc (ULONG)

Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM/2 driver for this command.

MCIERR_MISSING_FLAG
: Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
: The flags cannot be used together.

MCIERR_MISSING_STRING_ARGUMENT

Missing required string argument.

MCIERR_INVALID_ITEM_FLAG
Invalid item flag specified for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_OUTOFRANGE
Value given is out of range.

MCIERR_UNSUPPORTED_FUNCTION
Function not supported.

MCI_SET - Remarks

The parameters and flags for this message vary according to the selected device.

If the amp-mixer device does not support hardware mixing, MCI_UNSUPPORTED_FUNCTION will be returned.

MCI_SET - Related Messages

- MCI_STATUS

MCI_SET - Example Code

The following code illustrates setting the volume level for a device.

```
USHORT             usDeviceID;
MCI_SET_PARMS      msp;

msp.ulLevel = 50;                  /* 50% of volume */
msp.ulAudio = MCI_SET_AUDIO_ALL;
mciSendCommand(usDeviceID,
               MCI_SET,
               MCI_WAIT | MCI_SET_AUDIO |
               MCI_SET_VOLUME
               (PVOID) &msp, 0);
```

The following example illustrates how an application can set a particular connector's volume setting.

```
 MCI_AMP_SET_PARMS  mSet;
/* Set the volume of a particular connector. */
  mSet.ulValue = MCI_AMP_STREAM_CONNECTOR;
  mSet.ulLevel = 100;
  mSet.ulItem = MCI_AMP_SET_AUDIO;
  mSet.ulAudio = MCI_AMP_SET_BASS;

  ulError = mciSendCommand((USHORT)hMixer,
                 MCI_SET,
                 MCI_WAIT | MCI_SET_ITEM
                 (PVOID)&mSet,
                 0);
  if(ULONG_LOWD(ulError) != MCIERR_SUCCESS)
    {
}
```

MCI_SET - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_SET_CUEPOINT

MCI_SET_CUEPOINT Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_CUEPOINT_ON
This flag is used to set a cue point at the location specified in the *ulCuepoint* field of the MCI_CUEPOINT_PARMS data structure. The value in the *ulCuepoint* field is in the current time format.

Note: You can set only one cue point ON or OFF at a time.

MCI_SET_CUEPOINT_OFF
This flag is used to remove a cue point at the location specified in the *ulCuepoint* field of the MCI_CUEPOINT_PARMS data structure. The location specified must exactly match the location of the previously set cue point.

Note: You can set only one cue point ON or OFF at a time.

MCI_SET_CUEPOINT Parameter - pParam2

pParam2 (PMCI_CUEPOINT_PARMS)
Pointer to MCI_CUEPOINT_PARMS data structure.

MCI_SET_CUEPOINT Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	If the function succeeds, 0 is returned.
MCIERR_INVALID_DEVICE_ID	The device ID is not valid.
MCIERR_INSTANCE_INACTIVE	The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.
MCIERR_MISSING_FLAG	A required flag is missing.
MCIERR_UNSUPPORTED_FLAG	Given flag is unsupported for this device.
MCIERR_INVALID_CALLBACK_HANDLE	Given callback handle is invalid.
MCIERR_FILE_NOT_FOUND	File has not been loaded.
MCIERR_OUT_OF_MEMORY	Out of memory.
MCIERR_OUTOFRANGE	Units is out of range.
MCIERR_DUPLICATE_CUEPOINT	Given cue point already exists.
MCIERR_INVALID_CUEPOINT	Given cue point is invalid.
MCIERR_CUEPOINT_LIMIT_REACHED	The limit for cue points for this device has been reached. Delete one or more cue points and retry this message.
MCIERR_MISSING_PARAMETER	Required parameter is missing.

MCI_SET_CUEPOINT - Description

This message is used to set run-time cue points in the media device. The *ulCuepoint* field is in the current time format, but the cue-point notification messages are returned in MMTIME format.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure

pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_CUEPOINT_ON
: This flag is used to set a cue point at the location specified in the *ulCuepoint* field of the MCI_CUEPOINT_PARMS data structure. The value in the *ulCuepoint* field is in the current time format.

  Note: You can set only one cue point ON or OFF at a time.

MCI_SET_CUEPOINT_OFF
: This flag is used to remove a cue point at the location specified in the *ulCuepoint* field of the MCI_CUEPOINT_PARMS data structure. The location specified must exactly match the location of the previously set cue point.

  Note: You can set only one cue point ON or OFF at a time.

pParam2 (PMCI_CUEPOINT_PARMS)
Pointer to MCI_CUEPOINT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_OUT_OF_MEMORY
Out of memory.

MCIERR_OUTOFRANGE
Units is out of range.

MCIERR_DUPLICATE_CUEPOINT
Given cue point already exists.

MCIERR_INVALID_CUEPOINT
Given cue point is invalid.

MCIERR_CUEPOINT_LIMIT_REACHED
The limit for cue points for this device has been reached. Delete one or more cue points and retry this message.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SET_CUEPOINT - Remarks

When the device reaches the specified points during playback, the MM_MCICUEPOINT message is returned to the application using the window handle specified in the *hwndCallback* field. When setting a cue point on, the *hwndCallback* field *must* contain a valid window handle. An error is returned if a NULL or invalid window handle is specified in *pParam2*. Each cue point can be directed to a different window handle.

Only one cue point can be set at any given location in the media.

Cue points can only be set when a device element is loaded, and are reset when a new device element is loaded.

Cue points are persistent, that is they remain set after they are encountered. A cue point is only considered to have been encountered when the device passes the cue point location during playback or recording, not during seek operations.

If the length of a file cannot be determined, MCIERR_SUCCESS might be returned even though the specified point is out of range.

Devices that do not perform their own event detection might have less accurate cue points.

MCI_SET_CUEPOINT - Default Processing

As a general default, media drivers should support at least twenty cue points. If the number of supported cue points is exceeded, then MCIERR_CUEPOINT_LIMIT_REACHED will be returned.

MCI_SET_CUEPOINT - Example Code

The following code illustrates how to set run-time cue points for a media device.

```
                            /* Set a cue point 30 secs in the media */

   USHORT usDeviceID;
   HWND   hwndMyWindow;
   MCI_CUEPOINT_PARMS cuepointparms;        /* Cue point parameter
                                               structure             */

   /* Assign hwndCallback the handle to the PM Window - this returns
      MM_MCICUEPOINT messages.                                       */

   cuepointparms.hwndCallback = hwndMyWindow;
   cuepointparms.ulCuepoint = (ULONG) 30000; /* Current time format
                                               format = MS           */

   mciSendCommand( usDeviceID,        /* Device ID              */
    MCI_SET_CUEPOINT,                 /* MCI set cue point message */
    MCI_SET_CUEPOINT_ON | MCI_WAIT,
                                      /* Flags for this message   */
    (ULONG) &cuepointparms,           /* Data structure           */
    0);                               /* No user parm             */
```

MCI_SET_CUEPOINT - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Example Code
Glossary

MCI_SETIMAGEBUFFER

MCI_SETIMAGEBUFFER Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY	A notification message will be posted to the window specified in the *hwndCallback* field of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.
MCI_WAIT	Control is not to be returned until the action indicated by this message is completed or an error occurs.
MCI_CONVERT	This flag specifies that image format conversion will be performed. Data is assumed to be in the device-specific format. If MCI_CONVERT is specified, the data must be in the OS/2 uncompressed bit-map format.

MCI_SETIMAGEBUFFER Parameter - pParam2

pParam2 (PMCI_IMAGE_PARMS)
A pointer to the MCI_IMAGE_PARMS data structure.

MCI_SETIMAGEBUFFER Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM/2 driver for this command.

MCIERR_INSTANCE_INACTIVE
: Instance inactive.

MCIERR_INVALID_BUFFER
: Invalid return buffer given.

MCIERR_INVALID_BUFFER
: Invalid return buffer given.

MCIERR_FILE_NOT_FOUND
: File not found.

MCIERR_TARGET_DEVICE_FULL
: Target device is full.

MCI_SETIMAGEBUFFER - Description

This message writes data to the image capture buffer. The fields in the MCI_IMAGE_PARMS structure are used to interpret the data.

Using this message invalidates (or resets) the current element name or element HMMIO handle, since the element has been replaced by data from the application.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* field of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_CONVERT
: This flag specifies that image format conversion will be performed. Data is assumed to be in the device-specific format.

: If MCI_CONVERT is specified, the data must be in the OS/2 uncompressed bit-map format.

pParam2 (PMCI_IMAGE_PARMS)
A pointer to the MCI_IMAGE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER

Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM/2 driver for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_FILE_NOT_FOUND
File not found.

MCIERR_TARGET_DEVICE_FULL
Target device is full.

MCI_SETIMAGEBUFFER - Remarks

The format of the image data to be set is specified by the *ulPelFormat* and *usBitCount* fields of the MCI_IMAGE_PARMS data structure. If MCI_CONVERT is specified, the data must be in OS/2 bit-map format and will be converted to the device-specific format. The driver expects a BITMAPINFOHEADER2 data structure at the beginning of the buffer, followed by any palette data, and then the pel data. If MCI_CONVERT is not specified, the data will be placed directly into the device element buffer. If the current bits-per-pel, pixel-format or MCI_CONVERT values conflict, the message will fail.

On dual-plane image capture hardware devices, the image layer content is assumed. Output is clipped as needed to the visible regions of the display window.

Conversion from OS/2 bit-map format to YUVB format is accomplished with an I/O procedure which can use disk space for temporary storage. Therefore, it is possible that errors such as MCIERR_TARGET_DEVICE_FULL can occur.

MCI_SETIMAGEBUFFER - Example Code

The following code illustrates how to write data to the image capture buffer.

```
USHORT  usUserParm = 0;
ULONG   ulReturn;
BITMAPINFOHEADER2 *pbmphdr;
MMOTIONHEADER *pmmothdr;
MCI_IMAGE_PARMS mciImageParms;

memset (mciImageParms, 0x00, sizeof (MCI_IMAGE_PARMS));
mciImageParms.hwndCallback = hwndNotify;

/* If you desire to set from a standard format converted */
/* buffer                                                */
if (ulFlags & MCI_CONVERT)
     {
   /*******************************/
   /* For RGB BITMAP data buffer */
   /*******************************/
   mciImageParms.ulPelFormat = MCI_IMG_RGB;
   mciImageParms.usBitCount = 24;
   mciImageParms.ulImageCompression = MCI_IMG_COMP_NONE;
```

```
     mciImageParms.ulPelBufferWidth  = 200;
     mciImageParms.ulPelBufferHeight = 100;
     mciImageParms.ulBufLen = ((200 * 3) * 100) + sizeof
      (BITMAPINFOHEADER2);
     DosAllocMem (&mciImageParms.pPelBuffer,
                  mciImageParms.ulBufLen,
                  PAG_COMMIT | PAG_WRITE | PAG_READ);

     /* Set the BITMAP HEADER section to look like a real bitmap*/
     pbmphdr = (BITMAPINFOHEADER2 *)mciImageParms.pPelBuffer;
     pbmphdr->cbFix     = sizeof (BITMAPINFOHEADER2);
     pbmphdr->cx        = mciImageParms.ulPelBufferWidth;
     pbmphdr->cy        = mciImageParms.ulPelBufferHeight;
     pbmphdr->cPlanes   = 1;
     pbmphdr->cBitCount = mciImageParms.usBitCount;


     /* Set the BITMAP DATA section to RGB white. */
     memset ((PVOID)((LONG)mciImageParms.pPelBuffer + sizeof
     (BITMAPINFOHEADER2)
     ),
        0xFF, mciImageParms.ulBufLen - sizeof (BITMAPINFOHEADER2));
     }
    else
       {

     /********************************/
     /* For M-Motion YUV data buffer */
     /********************************/
     mciImageParms.ulPelFormat = MCI_IMG_YUV;
     mciImageParms.usBitCount = 12;
     mciImageParms.ulImageCompression = MCI_IMG_COMP_NONE;
     mciImageParms.ulPelBufferWidth  = 200;
     mciImageParms.ulPelBufferHeight = 100;
     mciImageParms.ulBufLen = (200 * 100) + ((200 * 100) >> 1) + sizeof
  (MMOTIONHEADER);
     DosAllocMem (&mciImageParms.pPelBuffer,
                mciImageParms.ulBufLen,
                  PAG_COMMIT | PAG_WRITE | PAG_READ);

     /* Set the BITMAP HEADER section to look like a real bitmap */
     pmmothdr = (MMOTIONHEADER *)mciImageParms.pPelBuffer;
     strncpy (&pmmothdr->mmID[0], "YUV12C", 6);
     pmmothdr->mmXlen = mciImageParms.ulPelBufferWidth ;
     pmmothdr->mmYlen = mciImageParms.ulPelBufferHeight;

     /* Leave the yuv buffer black for this example.  */
     }

  ulReturn = mciSendCommand(usDeviceID, MCI_SETIMAGEBUFFER,
                   MCI_WAIT | ulFlags,
                   (PVOID)&mciImageParms,
                   usUserParm);
```

MCI_SETIMAGEBUFFER - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_SETIMAGEPALETTE

MCI_SETIMAGEPALETTE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_REGISTERED
This flag sets the registered palette specified in the *usRegisteredMap* field of the MCI_PALETTE_PARMS data structure.

MCI_SETIMAGEPALETTE Parameter - pParam2

pParam2 (PMCI_PALETTE_PARMS)
A pointer to the MCI_PALETTE_PARMS data structure.

MCI_SETIMAGEPALETTE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds.

MCI_SETIMAGEPALETTE - Description

This message sets the palette or color map that is to be used for images loaded with subsequent MCI_SETIMAGEBUFFER messages.

This message does not affect motion video, an image that is already displayed, or images loaded via the MCI_RESTORE message.

This message applies only to palettized images.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_REGISTERED
This flag sets the registered palette specified in the *usRegisteredMap* field of the MCI_PALETTE_PARMS data structure.

pParam2 (PMCI_PALETTE_PARMS)
A pointer to the MCI_PALETTE_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds.

MCI_SETIMAGEPALETTE - Remarks

The map can either be a registered map or a map specified by the application.

If the number of palette entries in MCI_SETIMAGEPALETTE does not match the number of colors in the subsequent MCI_SETIMAGEBUFFER message, the image might be displayed incorrectly.

MCI_SETIMAGEPALETTE - Default Processing

Each image device will possess some default palette (or palettes) that will be used in palettized modes of operation. These defaults may be the current system palette.

MCI_SETIMAGEPALETTE - Topics

Select an item:
Description
Returns
Remarks
Default Processing
Glossary

MCI_SET_POSITION_ADVISE

MCI_SET_POSITION_ADVISE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_POSITION_ADVISE_ON
This flag specifies that position-change advise message frequency is to be set to the value specified in the *ulUnits* field of the MCI_POSITION_PARMS data structure.

MCI_SET_POSITION_ADVISE_OFF
This flag disables position-change advise messages.

MCI_SET_POSITION_ADVISE Parameter - pParam2

pParam2 (PMCI_POSITION_PARMS)
A pointer to the MCI_POSITION_PARMS data structure.

MCI_SET_POSITION_ADVISE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_INVALID_FLAG
Given flag is invalid.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_FILE_NOT_FOUND
File has not been loaded.

MCIERR_OUT_OF_MEMORY
Out of memory.

MCIERR_OUTOFRANGE
Unit is out of range.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SET_POSITION_ADVISE - Description

This message is used to set periodic position-change messages from the media device. The *ulUnits* field of the MCI_POSITION_PARMS data structure contains the interval that these messages are to be generated. The interval is relative to position 0 of the media. The *ulUnits* field is in the current time format, but the position-change messages are returned in MMTIME format.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_POSITION_ADVISE_ON
This flag specifies that position-change advise message frequency is to be set to the value specified in the *ulUnits* field of the MCI_POSITION_PARMS data structure.

MCI_SET_POSITION_ADVISE_OFF
This flag disables position-change advise messages.

pParam2 (PMCI_POSITION_PARMS)
A pointer to the MCI_POSITION_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_INVALID_FLAG
Given flag is invalid.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_FILE_NOT_FOUND

File has not been loaded.

MCIERR_OUT_OF_MEMORY
Out of memory.

MCIERR_OUTOFRANGE
Unit is out of range.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SET_POSITION_ADVISE - Remarks

Setting position-change advise causes MM_MCIPOSITIONCHANGE messages to be returned to the application whenever the specified media position is reached. The window handle specified in the *hwndCallback* field of MCI_POSITION_PARMS receives the MM_MCIPOSITIONCHANGE messages. When setting position advise on, a valid window handle must be specified in the *hwndCallback* field. An error is returned if a NULL or invalid window handle is specified.

Only one position-change advise message frequency can be specified; that is, setting a new frequency for position-change advise messages replaces the previously set position-change advise request.

A device element must be loaded before position advise can be set, and is reset when a new device element is loaded. Devices that do not perform their own event detection might have less accurate position-advise events.

Position advise messages are only generated during playback or recording, not during seek operations.

If MCI_SET_POSITION_ADVISE_OFF is specified *ulUnits* is ignored; otherwise, if the *ulUnits* field contains 0, the error MCIERR_OUTOFRANGE is returned.

If the length of a file cannot be determined, MCIERR_SUCCESS might be returned if *ulUnits* is out of range, and no MM_MCIPOSITIONCHANGE messages are generated.

MCI_SET_POSITION_ADVISE - Example Code

The following code illustrates how to set periodic position-change messages from a media device.

```
/* Request position advise notification every 2 seconds             */

USHORT usDeviceID;
HWND hwndMyWindow;
MCI_POSITION_PARMS positionparms; /* Position advise parm structure */

/* Assign hwndCallback the handle to the PM Window - this is where
   MM_MCIPOSITIONCHANGE messages will be received.                  */

positionparms.hwndCallback = hwndMyWindow;
positionparms.ulUnits = (ULONG) 2000; /* (Current time format = MS) */

mciSendCommand(usDeviceID,            /* Device ID                  */
 MCI_SET_POSITION_ADVISE,             /* MCI set position advise
                                         message                    */
 MCI_SET_POSITION_ADVISE_ON | MCI_WAIT,
                                      /* Flags for this message     */
 (PVOID) &positionparms,              /* Data structure             */
 0);                                  /* No user parm               */
```

MCI_SET_POSITION_ADVISE - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_SET_SYNC_OFFSET

MCI_SET_SYNC_OFFSET Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SET_SYNC_OFFSET Parameter - pParam2

pParam2 (PMCI_SYNC_OFFSET_PARMS)
A pointer to the MCI_SYNC_OFFSET_PARMS data structure.

MCI_SET_SYNC_OFFSET Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG

A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_OUT_OF_MEMORY
Out of memory.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SET_SYNC_OFFSET - Description

This message is used to specify positional offsets for devices operating in synchronization.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_SYNC_OFFSET_PARMS)
A pointer to the MCI_SYNC_OFFSET_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_OUT_OF_MEMORY
Out of memory.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_SET_SYNC_OFFSET - Remarks

This message sets the synchronization offset for a device. When MCI_PLAY or MCI_SEEK messages are sent to a synchronized device group, the *from* position of the play for each device is biased by its synchronization offset. The synchronization offset is specified in the currently set device units and is 0 by default.

MCI_SET_SYNC_OFFSET - Example Code

The following code illustrates how to specify positional offsets for operating synchronized devices.

```
                   /* Set the sync offset for the device to 10 secs */

USHORT usDeviceID;
MCI_SYNC_OFFSET_PARMS msoparms;

msoparms.ulOffset = (ULONG) 10000; /* Current time format = MS      */

mciSendCommand( usDeviceID,        /* Device ID                     */
 MCI_SET_SYNC_OFFSET,              /* MCI set sync offset message   */
 MCI_WAIT,                         /* Flag for this message         */
 (PVOID) &msoparms,                /* Data structure                */
 0);                               /* No user parm                  */
```

MCI_SET_SYNC_OFFSET - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_SETTUNER

MCI_SETTUNER Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain the following flags

MCI_NOTIFY	Posts a notification message to the window specified in the *hwndCallback* parameter of the data structure identified by *ulParam2* when the action indicated by this message is completed.
MCI_WAIT	Does not return control until the action indicated by this message is completed.
MCI_DGV_FREQUENCY	Sets the frequency being sent to the device driver to the value in the *ulFrequency* field of the MCI_DGV_TUNER_PARMS structure. Overrides channel, region, and fine-tuning.
MCI_DGV_TV_CHANNEL	Sets the channel to the value in the *ulTVChannel* field of the MCI_DGV_TUNER_PARMS structure. Channel is used along with region and fine-tuning to calculate the frequency.
MCI_DGV_REGION	Sets the channel to the value in the *pszRegion* field of the MCI_DGV_TUNER_PARMS structure. Region is used along with channel and fine-tuning to calculate the frequency.
MCI_DGV_FINETUNE_PLUS	Indicates that the value in the *lFineTune* field of the MCI_DGV_TUNER_PARMS structure is positive. Fine-tuning is used along with region and channel to calculate the frequency.
MCI_DGV_FINETUNE_MINUS	Indicates that the value in the *lFineTune* field of the MCI_DGV_TUNER_PARMS structure is negative. In other words, multiply the value in *lFineTune* by -1.

MCI_SETTUNER Parameter - pParam2

pParam2 (PMCI_DGV_TUNER_PARMS)
A pointer to an MCI_DGV_TUNER_PARMS structure.

MCI_SETTUNER Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS

MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_TUNER_NO_HW
Device has no tuner support.

MCIERR_TUNER_CHANNEL_SKIPPED
Channel skipped in region.

MCIERR_TUNER_CHANNEL_TOO_HIGH
Channel too high for region.

MCIERR_TUNER_CHANNEL_TOO_LOW
Channel too low for region.

MCIERR_AUD_CHANNEL_OUTOFRANGE
Audio channel out of range.

MCIERR_INVALID_REGION
Region file either does not exist or is invalid.

MCIERR_TUNER_REGION_NOT_SET
Region has not been set.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified.

MCIERR_MISSING_FLAG
Flag missing for this command.

MCIERR_UNSUPPORTED_FLAG
Flag not supported by this MMPM/2 driver for this command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_SETTUNER - Description

This message causes the digital video MCD to change the frequency that the tuner device is tuned to.

ulParam1 (ULONG)
This parameter can contain the following flags

MCI_NOTIFY
Posts a notification message to the window specified in the *hwndCallback* parameter of the data structure identified by *ulParam2* when the action indicated by this message is completed.

MCI_WAIT
Does not return control until the action indicated by this message is completed.

MCI_DGV_FREQUENCY

Sets the frequency being sent to the device driver to the value in the *ulFrequency* field of the MCI_DGV_TUNER_PARMS structure. Overrides channel, region, and fine-tuning.

MCI_DGV_TV_CHANNEL
: Sets the channel to the value in the *ulTVChannel* field of the MCI_DGV_TUNER_PARMS structure. Channel is used along with region and fine-tuning to calculate the frequency.

MCI_DGV_REGION
: Sets the channel to the value in the *pszRegion* field of the MCI_DGV_TUNER_PARMS structure. Region is used along with channel and fine-tuning to calculate the frequency.

MCI_DGV_FINETUNE_PLUS
: Indicates that the value in the *lFineTune* field of the MCI_DGV_TUNER_PARMS structure is positive. Fine-tuning is used along with region and channel to calculate the frequency.

MCI_DGV_FINETUNE_MINUS
: Indicates that the value in the *lFineTune* field of the MCI_DGV_TUNER_PARMS structure is negative. In other words, multiply the value in *lFineTune* by -1.

pParam2 (PMCI_DGV_TUNER_PARMS)
A pointer to an MCI_DGV_TUNER_PARMS structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_TUNER_NO_HW
: Device has no tuner support.

MCIERR_TUNER_CHANNEL_SKIPPED
: Channel skipped in region.

MCIERR_TUNER_CHANNEL_TOO_HIGH
: Channel too high for region.

MCIERR_TUNER_CHANNEL_TOO_LOW
: Channel too low for region.

MCIERR_AUD_CHANNEL_OUTOFRANGE
: Audio channel out of range.

MCIERR_INVALID_REGION
: Region file either does not exist or is invalid.

MCIERR_TUNER_REGION_NOT_SET
: Region has not been set.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified.

MCIERR_MISSING_FLAG
: Flag missing for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM/2 driver for this command.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCIERR_INSTANCE_INACTIVE
: Instance inactive.

MCI_SETTUNER - Remarks

- Tuner channels can be any positive number including 0. However, tuner channels are validated according to the region.
- Region can be any character string, but there must be a corresponding file (*character_string* .RGN) in the \MMOS2\REGION subdirectory. A partial list of regions includes:

USA.RGN	USA Air
USACATV.RGN	USA Cable
CCIR.RGN	Western Europe CCIR Air
CCIRCATV.RGN	Western Europe CCIR Cable
AUSTR.RGN	Australia
JAPAN.RGN	Japanese AIR
JAPANCATV.RGN	Japanese Cable

  New regions can be created, allowing one to expand the regions supported or to block out undesirable channels.
- MCI_DGV_FINETUNE_PLUS and MCI_DGV_FINETUNE_MINUS are mutually exclusive.
- Channel and region do not have to be set every time; values will be remembered. If finetuning is necessary, it will have to be reset every time the channel or region is reset.
- MCI_DGV_FREQUENCY temporarily overrides the currently set channel and region. The next MCI_SETTUNER command without MCI_DGV_FREQUENCY set will revert back to the previously set channel and region.
- If the region is set without a channel, the lowest channel available for that region will be used.

MCI_SETTUNER - Example Code

The following example shows how to set the frequency for the tuner device using MCI_SETTUNER.

```
USHORT               usDeviceID;
MCI_DGV_TUNER_PARMS  settuner;

settuner.usDeviceID  = usDeviceID; /* Device ID */
settuner.ulFrequency = 24725; /* Frequency for channel 29 in USA Cable TV */
settuner.pszRegion   = NULL;  /* Region, Channel and Finetune are not */
settuner.usTVChannel = 0;     /* needed since we are inputting the    */   */
settuner.lFineTune   = 0;     /* frequency.                           */   */


ulError = mciSendCommand ( usDeviceID,
                           MCI_SETTUNER,
                           MCI_WAIT | MCI_DGV_FREQUENCY,
                           &settuner,
                           0);
```

MCI_SETTUNER - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_SPIN

MCI_SPIN Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags: The MCI_SPIN_UP and MCI_SPIN_DOWN flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SPIN_UP
This flag starts the disc spinning.

MCI_SPIN_DOWN
This flag stops the disc from spinning.

MCI_SPIN Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_SPIN Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID

The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE message to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_SPIN - Description

This message is sent to spin the player up or down.

ulParam1 (ULONG)
This parameter can contain any of the following flags: The MCI_SPIN_UP and MCI_SPIN_DOWN flags are mutually exclusive.

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SPIN_UP
This flag starts the disc spinning.

MCI_SPIN_DOWN
This flag stops the disc from spinning.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE message to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCI_SPIN - Default Processing

The MCI_SPIN_UP flag is assumed by default.

MCI_SPIN - Example Code

The following code illustrates how to start a videodisc spinning and request notification upon completion.

```
/* Start the videodisc spinning, requesting notification of
completion                                                    */

USHORT usDeviceID;
HWND hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms; /* Generic message parms
                                            structure              */

                 /* Assign hwndCallback the handle to the PM Window */

mciGenericParms.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,           /* Device ID                */
 MCI_SPIN,                            /* MCI spin message         */
 MCI_NOTIFY | MCI_SPIN_UP,            /* Flags for this message   */
 (PVOID) &mciGenericParms,            /* Data structure           */
 0 );                                 /* No user parm             */
```

MCI_SPIN - Topics

Select an item:
Description
Returns
Default Processing
Example Code
Glossary

MCI_STATUS

MCI_STATUS Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_STATUS_START
Returns the starting position of the media. Specify MCI_STATUS_POSITION as the status item in *ulItem* .

MCI_TRACK
A status track parameter is included in the *ulTrack* field of the data structure pointed to by *pParam2* . If MCI_TRACK is specified, the status item must be either MCI_STATUS_POSITION or MCI_STATUS_LENGTH. When used with MCI_STATUS_POSITION, the starting position of the given track, segment, or chapter is returned. When used with MCI_STATUS_LENGTH, the length of the given track, segment, element, or chapter is returned.

MCI_STATUS_CONNECTOR
If this flag is specified, a valid connector must be in the *ulValue* field of MCI_STATUS_PARMS. The specific audio setting to be queried is set in the *ulItem* field. MCI_STATUS_CONNECTOR and MCI_STATUS_ITEM are mutually exclusive. If both of these flags are specified, MCIERR_INVALID_FLAG will be returned.

MCI_STATUS_ITEM
Indicates that the *ulItem* field of the data structure identified by *pParam2* contains a constant specifying the status item in question. The following constants are defined:

MCI_STATUS_AUDIO
One of the following status audio parameters must be included in the *ulValue* field of the data structure pointed to by *pParam2* . The following predefined channel numbers can be specified. You can specify other channel numbers by specifying the appropriate channel number.

MCI_STATUS_AUDIO_ALL
Returns MCI_TRUE if all channels are on; otherwise, returns MCI_FALSE. This is the default value.

MCI_STATUS_AUDIO_LEFT
Returns MCI_TRUE if the left channel is on; otherwise, returns MCI_FALSE.

MCI_STATUS_AUDIO_RIGHT
Returns MCI_TRUE if the right channel is on; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_PASTE
Returns MCI_TRUE if compatible data is to be placed in clipboard; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_REDO
Returns MCI_TRUE if an operation that was undone can be redone; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_UNDO
Returns MCI_TRUE if a change has been made that can be undone; otherwise, returns MCI_FALSE.

MCI_STATUS_CLIPBOARD
Returns MCI_TRUE if the clipboard contains information understood by the current device; otherwise returns MCI_FALSE.

MCI_STATUS_CURRENT_TRACK
Returns the current track, segment, or chapter number.

MCI_STATUS_LENGTH
Returns the total media length in units as specified in the MCI_SET message with the MCI_SET_TIME_FORMAT flag.

Note: If the time format has been set to MCI_FORMAT_TMSF, the actual time value returned will be in the format MCI_FORMAT_MSF.

If the media length cannot be determined because a playlist is currently loaded, or for any other reason, MCIERR_INDETERMINATE_LENGTH is returned.

MCI_STATUS_MODE
Returns the current mode of the device. Possible values are:

- MCI_MODE_NOT_READY
- MCI_MODE_PAUSE
- MCI_MODE_PLAY
- MCI_MODE_STOP
- MCI_MODE_RECORD
- MCI_MODE_SEEK

MCI_STATUS_MEDIA_PRESENT
Returns MCI_TRUE or MCI_FALSE. If the device does not have removable media, it returns MCI_TRUE. Note that this function is only applicable to devices which are dependent on removable media. Receiving a return of MCI_FALSE indicates that the device cannot function without inserting the media into the device. Examples of devices which might return MCI_FALSE to this command are CD audio and videodisc devices.

MCI_STATUS_MONITOR
Returns MCI_ON or MCI_OFF to indicate whether monitoring of the incoming video signal is turned on or off.

MCI_STATUS_NUMBER_OF_TRACKS
Returns the total number of playable tracks, segments, or chapters.

MCI_STATUS_POSITION
Returns the current position.

MCI_STATUS_POSITION_IN_TRACK
Returns the current position relative to the beginning of the current track, segment, or chapter.

MCI_STATUS_READY
Returns MCI_TRUE if the device is ready; otherwise, returns MCI_FALSE.

MCI_STATUS_SPEED_FORMAT
Returns the currently set speed format. Possible values are:

- MCI_FORMAT_PERCENTAGE
- MCI_FORMAT_FPS

MCI_STATUS_TIME_FORMAT
Returns the currently set time format. Possible values are:

- MCI_FORMAT_MILLISECONDS
- MCI_FORMAT_MMTIME
- MCI_FORMAT_MSF
- MCI_FORMAT_TMSF
- MCI_FORMAT_CHAPTERS
- MCI_FORMAT_FRAMES
- MCI_FORMAT_HMS
- MCI_FORMAT_TRACKS
- MCI_FORMAT_BYTES
- MCI_FORMAT_SAMPLES

- MCI_FORMAT_HMSF
- MCI_FORMAT_SET_SMPTE_24
- MCI_FORMAT_SET_SMPTE_25
- MCI_FORMAT_SET_SMPTE_30
- MCI_FORMAT_SET_SMPTE_30DROP
- MCI_FORMAT_SET_SONGPTR

MCI_STATUS_VIDEO
Returns MCI_TRUE if video is on; otherwise returns MCI_FALSE.

MCI_STATUS_VOLUME
Returns the actual volume level set in the device as a percentage of the maximum achievable effect. The left channel is returned in the low-order word, and the right channel is returned in the high-order word.

Amplifier Mixer Extensions

The following additional status items apply to amplifier-mixer devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_AMP_STATUS_BALANCE
Returns a balance level for this mixer channel. A value of zero indicates full left balance while 100 indicates full right balance, and 50 indicates neutral balance.

MCI_AMP_STATUS_BASS
Returns a bass level for this mixer channel as a percentage of the maximum achievable bass effect.

MCI_AMP_STATUS_GAIN
Returns the gain setting as a percentage of the maximum achievable effect.

MCI_AMP_STATUS_PITCH
Returns the pitch as a percentage of the maximum achievable effect.

MCI_AMP_STATUS_TREBLE
Returns treble level for this mixer channel as a percentage of the maximum treble effect.

If MCI_STATUS_CONNECTOR is specified, the following additional items can be specified in the *ulItem* field of MCI_STATUS_PARMS.

MCI_AMP_STATUS_ALC
Returns the current auto-level control setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_BASS
Returns the current bass setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_BALANCE
Returns the current balance setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CHORUS
Returns the current chorus setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CROSSOVER
Returns the current crossover setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM1
Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM2
Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect.

MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM3
Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_GAIN
Returns the current gain setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_LOUDNESS
Returns the current loudness setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MID
Returns the current mid setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MONITOR
Returns the current monitor setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MUTE
Returns the current mute setting for the connector specified in *ulValue* of MCI_STATUS_PARMS. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_PITCH
Returns the current pitch setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_REVERB
Returns the current reverb setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_STEREOENHANCE
Returns the current stereo enhance setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_TREBLE
Returns the current treble setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_VOLUME
Returns the current volume setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

CD Audio Extensions

The following additional status items apply to CD audio devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_CD_STATUS_TRACK_TYPE
Returns one of the following:

- MCI_CD_TRACK_AUDIO
- MCI_CD_TRACK_DATA
- MCI_CD_TRACK_OTHER

MCI_CD_STATUS_TRACK_COPYPERMITTED
Returns MCI_TRUE if digital copying is permitted; otherwise, returns MCI_FALSE.

MCI_CD_STATUS_TRACK_CHANNELS
Returns the number of audio channels on the track.

MCI_CD_STATUS_TRACK_PREEMPHASIS
Returns MCI_TRUE if the track was recorded with pre-emphasis; otherwise, returns MCI_FALSE.

Note: When used with the MCI_TRACK flag, these items return the status information of the specified track instead of the current track.

CD/XA Extensions

The following extensions apply to CD-XA devices and can be specified for the *ulItem* field of the data structure pointed to by *pParam2*:

MCI_CDXA_STATUS_CHANNEL
Returns the destination of the data in channel *ulChannel*. Returns one of the following:

- MCI_CDXA_AUDIO_DEVICE
- MCI_CDXA_AUDIO_BUFFER
- MCI_CDXA_VIDEO_BUFFER
- MCI_CDXA_DATA_BUFFER
- MCI_CDXA_NONE

Digital Video Extensions

The following additional status items apply to digital video devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag.

MCI_DGV_STATUS_FORMATTAG
Returns WAVE_FORMAT_PCM, the only format currently supported by the digital video device. If a movie is loaded that contains a format other than PCM, the format used in the movie will be returned.

MCI_DGV_STATUS_DROPPED_FRAME_PCT
Returns the percentage of dropped frames for recording or playback operations. The value returned is in the range 0-100, where a value of zero indicates that no frame drops are occurring or have occurred and a value of 100 indicates that all frames are being dropped or have been dropped. This status value can be queried during a recording operation to obtain the cumulative percentage of frame drops that have occurred since recording began, or during playback to obtain the cumulative percentage of frame drops that have occurred since playback began or was resumed after a seek, pause, or stop. If the value is queried when the device is stopped, the percentage of dropped frames accumulated at the end of the last playback or recording operation that was performed is returned. A value of zero is returned if no playback or recording operations have been performed, the device is seeking or has been seeked, the device is paused or stopped, or the device is playing in scan mode.

MCI_DGV_STATUS_SAMPLESPERSEC
Returns the currently set samples per second used for playing, recording, and saving.

MCI_DGV_STATUS_BITSPERSAMPLE
Returns the currently set bits per sample used for playing, recording, and saving.

MCI_DGV_STATUS_CHANNELS
Returns the currently set number of channels used for playing, recording, and saving.

MCI_DGV_STATUS_HWND
Returns the handle of the playback window.

MCI_DGV_STATUS_VIDEO_COMPRESSION
Returns the current FOURCC compression format for recording of motion video. Only symmetric compressors will be enabled for real-time recording.

MCI_DGV_STATUS_VIDEO_QUALITY
Returns the currently set compression quality level for recording of motion video.

MCI_DGV_STATUS_MONITOR
Returns MCI_ON or MCI_OFF to indicate whether monitoring of the incoming video signal is on or off.

MCI_DGV_STATUS_HWND_MONITOR
Returns the monitor window handle.

MCI_DGV_STATUS_REF_INTERVAL
Returns the value of *n* where *n* refers to a reference frame being inserted every *n*th frame.

MCI_DGV_STATUS_IMAGE_BITSPERPEL
Returns the pel format used for saving bitmaps.

MCI_DGV_STATUS_IMAGE_PELFORMAT
Returns the data format used of image data for the capture device. Possible values are:

- MMIO_RGB_5_6_5

  Each pixel is represented by 16 bits of data as follows:

15:5	Red level in the range 0-31
10:6	Green level in the range 0-63
4:5	Blue level in the range 0-31

- MMIO_YUV_4_1_1

  This format uses 16 bits per pixel, but uses 4-pixel horizontal chrominance subsampling. Each pixel has a unique luminance value (Y) with a single chrominance value (U and V) shared by four pixels. Y, U, and V all have 7 bits of significance in this format.

23:8	Red level in the range 0-255
15:8	Green level in the range 0-255
7:8	Blue level in the range 0-255

- MMIO_YUV_4_2_2

  4 bytes of Y, 2 bytes of U, 2 bytes of V; all 8-bit values in this form YUYVYUYV

MCI_DGV_STATUS_FORWARD
Returns MCI_TRUE if playing forward; otherwise returns MCI_FALSE.

MCI_DGV_STATUS_NORMAL_RATE
Returns the normal-play rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second.

MCI_DGV_STATUS_VIDEO_X_EXTENT
Returns the horizontal (X) extent of the digital motion video image for the currently loaded motion video device element.

MCI_DGV_STATUS_VIDEO_Y_EXTENT
Returns the vertical (Y) extent of the digital motion video image for the currently loaded motion video device element.

MCI_DGV_STATUS_BRIGHTNESS
Returns the brightness level.

MCI_DGV_STATUS_CONTRAST
Returns the contrast level.

MCI_DGV_STATUS_HUE
Returns the hue level.

MCI_DGV_STATUS_SATURATION
Returns the saturation level.

MCI_DGV_STATUS_RECORD_AUDIO
Returns MCI_ON or MCI_OFF to indicate whether recording the audio soundtrack has been turned on or off.

MCI_DGV_STATUS_SPEED
Returns the digital video speed in frames per second.

MCI_DGV_STATUS_TRANSPARENT_COLOR
Returns a value representing the transparent color used as the chroma-key on video overlay hardware.

MCI_DGV_STATUS_VIDEO_RECORD_FRAME_DURATION
Returns the frame rate for recording as the time duration of each frame in microseconds.

MCI_DGV_STATUS_TUNER_TV_CHANNEL
This flag returns the channel that the tuner device is tuned to.

MCI_DGV_STATUS_TUNER_HIGH_TV_CHANNEL
This flag returns the highest channel for the region.

MCI_DGV_STATUS_TUNER_LOW_TV_CHANNEL
This flag returns the lowest channel for the region.

MCI_DGV_STATUS_TUNER_FINETUNE
This flag returns the fine-tuning value that the tuner device is tuned to.

MCI_DGV_STATUS_TUNER_FREQUENCY
This flag returns the frequency value that the tuner device is tuned to.

MCI_DGV_STATUS_VALID_SIGNAL
This flag returns MCI_TRUE if there is a signal present.

Sequencer Extensions

The following additional status items apply to MIDI sequencer devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_SEQ_STATUS_DIVTYPE
Returns one of the following values as the current division type of a sequence:

- MCI_SEQ_DIV_PPQN
- MCI_SEQ_DIV_SMPTE_24
- MCI_SEQ_DIV_SMPTE_25
- MCI_SEQ_DIV_SMPTE_25
- MCI_SEQ_DIV_SMPTE_30
- MCI_SEQ_DIV_SMPTE_30DROP

MCI_SEQ_STATUS_MASTER
Returns the synchronization type used for master operation.

MCI_SEQ_STATUS_OFFSET
Returns the current SMPTE offset of a sequence.

MCI_SEQ_STATUS_PORT
Returns the MIDI device ID for the current port used by the sequence.

MCI_SEQ_STATUS_SLAVE
Returns the synchronization type used for slave operation.

MCI_SEQ_STATUS_TEMPO
Returns the current tempo of a MIDI sequence in beats-per-minute for PPQN files, or frames-per-second for SMPTE files. Currently this function is not supported by the IBM sequencer.

Videodisc Extensions

The following additional status items apply to videodisc devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_VD_STATUS_SPEED
Returns the speed in the currently set speed format.

MCI_VD_STATUS_FORWARD
Returns MCI_TRUE if playing forward; otherwise, returns MCI_FALSE.

MCI_VD_MEDIA_TYPE
Returns one of the following:

- MCI_VD_MEDIA_CAV
- MCI_VD_MEDIA_CLV
- MCI_VD_MEDIA_OTHER

MCI_VD_STATUS_SIDE
Returns 1 or 2 to indicate which side of the disc is loaded.

MCI_VD_STATUS_DISC_SIZE
Returns the size of the loaded disc in inches (8 or 12).

Video Overlay Extensions

The following additional items apply to video overlay devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag.

MCI_OVLY_STATUS_HWND
Returns the handle of the playback window.

MCI_OVLY_STATUS_IMAGE_COMPRESSION
Returns the compression format of the currently loaded bitmap/image.

MCI_OVLY_STATUS_BITSPERPEL
Returns the number of bits per pel of the currently loaded bitmap/image. Return values include:

- MCI_IMG_PALETTE
- MCI_IMG_RGB
- MCI_IMG_YUV

MCI_OVLY_STATUS_PELFORMAT
Returns the pel format of the currently loaded bitmap/image.

MCI_OVLY_STATUS_BRIGHTNESS
Returns the brightness level.

MCI_OVLY_STATUS_CONTRAST
Returns the contrast level.

MCI_OVLY_STATUS_HUE
Returns the hue level.

MCI_OVLY_STATUS_SATURATION
Returns the saturation level.

MCI_OVLY_STATUS_SHARPNESS
Returns the sharpness level.

MCI_OVLY_STATUS_TRANSPARENT_COLOR
Returns a value representing the RGB value or palette value, which specifies the transparent color. RGB values are returned as a 32-bit RGB2 data item.

MCI_OVLY_STATUS_TRANSPARENT_TYPE
Returns a value representing information to assist in interpreting the MCI_OVLY_STATUS_TRANSPARENT_COLOR.

Return values include:

- MCI_IMG_PALETTE
- MCI_IMG_RGB
- MCI_IMG_YUV

MCI_OVLY_STATUS_GREYSCALE
Returns MCI_ON or MCI_OFF.

MCI_OVLY_STATUS_IMAGE_COMPRESSION
Returns the compression type for saving still images.

MCI_OVLY_STATUS_IMAGE_BITSPERPEL
Returns the number of bits per pel used for the image file to be saved.

MCI_OVLY_STATUS_IMAGE_PELFORMAT
Returns the pel format used for saving bitmaps.

MCI_OVLY_STATUS_IMAGE_QUALITY
Returns the quality of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_X_EXTENT
Returns the width, in pels, of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_Y_EXTENT
Returns the height, in pels, of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_FILE_FORMAT
Returns the format in which an image capture will be stored when saved.

Wave Audio Extensions

The following additional status items apply to wave audio devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_WAVE_STATUS_FORMATTAG
Returns the currently set format tag used for playing, recording, and saving.

MCI_WAVE_STATUS_CHANNELS
Returns the currently set channel count used for playing, recording, and saving.

MCI_WAVE_STATUS_SAMPLESPERSEC
Returns the currently set samples per second used for playing, recording, and saving.

MCI_WAVE_STATUS_AVGBYTESPERSEC
Returns the currently set bytes per second used for playing, recording, and saving. Playback software can use this number to estimate required buffer sizes. Refer to the RIFF WAVE format documentation for more information.

MCI_WAVE_STATUS_BLOCKALIGN
Returns the currently set block alignment used for playing, recording, and saving.

MCI_WAVE_STATUS_BITSPERSAMPLE
Returns the currently set bits per sample used for playing, recording, and saving.

MCI_WAVE_STATUS_LEVEL
Returns the current record or playback level. The value is returned as an 8-bit or 16-bit value, depending on the sample size being used. The right or Mono channel level is returned in the low-order word. The left channel level is returned in the high-order word.

MCI_STATUS Parameter - pParam2

pParam2 (PMCI_STATUS_PARMS)
A pointer to the MCI_STATUS_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_CDXA_STATUS_PARMS
A pointer to the MCI_CDXA_STATUS_PARMS data structure.

MCI_STATUS Return Value - rc

rc (ULONG)

Note: The format of the *ulReturn* value in this structure is defined by the high-order word of the value returned by mciSendCommand. This value is used by mciSendString to determine how to convert the *ulReturn* value to string form. For a list of the possible format values, see the MMDRVOS2.H header file. If the low-order word returned is MCIERR_SUCCESS, the high-order word could be other errors or a value. A returned value defines the format of *ulReturn* as defined in MMDRVOS2.H. For example, 0x5000 = MCI_TRUE_FALSE_RETURN.

Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
: Invalid flag specified for this command.

MCIERR_UNSUPPORTED_FLAG
: Flag not supported by this MMPM/2 driver for this command.

MCIERR_MISSING_FLAG
: Flag missing for this MMPM/2 command.

MCIERR_UNSUPPORTED_FUNCTION
: Function not supported by the media driver being used.

MCIERR_INVALID_ITEM_FLAG
: Invalid item flag specified for this command.

MCIERR_TUNER_NO_HW
: Device has no tuner support.

MCIERR_TUNER_MODE
: Frequency was last set directly. MCI_DGV_STATUS_TUNER_TV_CHANNEL and MCI_DGV_STATUS_TUNER_FINETUNE cannot be used. Use MCI_DGV_STATUS_FREQUENCY.

MCIERR_SIGNAL_INVALID
: No valid signal present.

MCI_STATUS - Description

This message is used to obtain information about the status of a device instance. MCI_STATUS returns the values most recently *set* by MCI_SET, MCI_LOAD, MCI_SETTUNER, and MCI_SETIMAGEBUFFER operations.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_STATUS_START
: Returns the starting position of the media. Specify MCI_STATUS_POSITION as the status item in *ulItem* .

MCI_TRACK
: A status track parameter is included in the *ulTrack* field of the data structure pointed to by *pParam2* . If MCI_TRACK is specified, the status item must be either MCI_STATUS_POSITION or MCI_STATUS_LENGTH. When used with MCI_STATUS_POSITION, the starting position of the given track, segment, or chapter is returned. When used with MCI_STATUS_LENGTH, the length of the given track, segment, element, or chapter is returned.

MCI_STATUS_CONNECTOR
: If this flag is specified, a valid connector must be in the *ulValue* field of MCI_STATUS_PARMS. The specific audio

setting to be queried is set in the *ulItem* field. MCI_STATUS_CONNECTOR and MCI_STATUS_ITEM are mutually exclusive. If both of these flags are specified, MCIERR_INVALID_FLAG will be returned.

MCI_STATUS_ITEM

Indicates that the *ulItem* field of the data structure identified by *pParam2* contains a constant specifying the status item in question. The following constants are defined:

MCI_STATUS_AUDIO

One of the following status audio parameters must be included in the *ulValue* field of the data structure pointed to by *pParam2* . The following predefined channel numbers can be specified. You can specify other channel numbers by specifying the appropriate channel number.

MCI_STATUS_AUDIO_ALL

Returns MCI_TRUE if all channels are on; otherwise, returns MCI_FALSE. This is the default value.

MCI_STATUS_AUDIO_LEFT

Returns MCI_TRUE if the left channel is on; otherwise, returns MCI_FALSE.

MCI_STATUS_AUDIO_RIGHT

Returns MCI_TRUE if the right channel is on; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_PASTE

Returns MCI_TRUE if compatible data is to be placed in clipboard; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_REDO

Returns MCI_TRUE if an operation that was undone can be redone; otherwise, returns MCI_FALSE.

MCI_STATUS_CAN_UNDO

Returns MCI_TRUE if a change has been made that can be undone; otherwise, returns MCI_FALSE.

MCI_STATUS_CLIPBOARD

Returns MCI_TRUE if the clipboard contains information understood by the current device; otherwise returns MCI_FALSE.

MCI_STATUS_CURRENT_TRACK

Returns the current track, segment, or chapter number.

MCI_STATUS_LENGTH

Returns the total media length in units as specified in the MCI_SET message with the MCI_SET_TIME_FORMAT flag.

Note: If the time format has been set to MCI_FORMAT_TMSF, the actual time value returned will be in the format MCI_FORMAT_MSF.

If the media length cannot be determined because a playlist is currently loaded, or for any other reason, MCIERR_INDETERMINATE_LENGTH is returned.

MCI_STATUS_MODE

Returns the current mode of the device. Possible values are:

- MCI_MODE_NOT_READY
- MCI_MODE_PAUSE
- MCI_MODE_PLAY
- MCI_MODE_STOP
- MCI_MODE_RECORD
- MCI_MODE_SEEK

MCI_STATUS_MEDIA_PRESENT

Returns MCI_TRUE or MCI_FALSE. If the device does not have removable media, it returns MCI_TRUE. Note that this function is only applicable to devices which are dependent on removable media. Receiving a return of MCI_FALSE indicates that the device cannot function without inserting the media into the device. Examples of devices which might return MCI_FALSE to this command are CD audio and videodisc devices.

MCI_STATUS_MONITOR

Returns MCI_ON or MCI_OFF to indicate whether monitoring of the incoming video signal is turned on or off.

MCI_STATUS_NUMBER_OF_TRACKS

Returns the total number of playable tracks, segments, or chapters.

MCI_STATUS_POSITION
Returns the current position.

MCI_STATUS_POSITION_IN_TRACK
Returns the current position relative to the beginning of the current track, segment, or chapter.

MCI_STATUS_READY
Returns MCI_TRUE if the device is ready; otherwise, returns MCI_FALSE.

MCI_STATUS_SPEED_FORMAT
Returns the currently set speed format. Possible values are:

- MCI_FORMAT_PERCENTAGE
- MCI_FORMAT_FPS

MCI_STATUS_TIME_FORMAT
Returns the currently set time format. Possible values are:

- MCI_FORMAT_MILLISECONDS
- MCI_FORMAT_MMTIME
- MCI_FORMAT_MSF
- MCI_FORMAT_TMSF
- MCI_FORMAT_CHAPTERS
- MCI_FORMAT_FRAMES
- MCI_FORMAT_HMS
- MCI_FORMAT_TRACKS
- MCI_FORMAT_BYTES
- MCI_FORMAT_SAMPLES
- MCI_FORMAT_HMSF
- MCI_FORMAT_SET_SMPTE_24
- MCI_FORMAT_SET_SMPTE_25
- MCI_FORMAT_SET_SMPTE_30
- MCI_FORMAT_SET_SMPTE_30DROP
- MCI_FORMAT_SET_SONGPTR

MCI_STATUS_VIDEO
Returns MCI_TRUE if video is on; otherwise returns MCI_FALSE.

MCI_STATUS_VOLUME
Returns the actual volume level set in the device as a percentage of the maximum achievable effect. The left channel is returned in the low-order word, and the right channel is returned in the high-order word.

Amplifier Mixer Extensions

The following additional status items apply to amplifier-mixer devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_AMP_STATUS_BALANCE
Returns a balance level for this mixer channel. A value of zero indicates full left balance while 100 indicates full right balance, and 50 indicates neutral balance.

MCI_AMP_STATUS_BASS
Returns a bass level for this mixer channel as a percentage of the maximum achievable bass effect.

MCI_AMP_STATUS_GAIN
Returns the gain setting as a percentage of the maximum achievable effect.

MCI_AMP_STATUS_PITCH
Returns the pitch as a percentage of the maximum achievable effect.

MCI_AMP_STATUS_TREBLE
Returns treble level for this mixer channel as a percentage of the maximum treble effect.

If MCI_STATUS_CONNECTOR is specified, the following additional items can be specified in the *ulItem* field of MCI_STATUS_PARMS.

MCI_AMP_STATUS_ALC
Returns the current auto-level control setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect.

MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_BASS
: Returns the current bass setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_BALANCE
: Returns the current balance setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CHORUS
: Returns the current chorus setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CROSSOVER
: Returns the current crossover setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM1
: Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM2
: Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_CUSTOM3
: Returns the current custom effect setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_GAIN
: Returns the current gain setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_LOUDNESS
: Returns the current loudness setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MID
: Returns the current mid setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MONITOR
: Returns the current monitor setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_MUTE
: Returns the current mute setting for the connector specified in *ulValue* of MCI_STATUS_PARMS. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_PITCH
: Returns the current pitch setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_REVERB
: Returns the current reverb setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_STEREOENHANCE
: Returns the current stereo enhance setting for the connector specified in *ulValue* of

MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_TREBLE
Returns the current treble setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

MCI_AMP_STATUS_VOLUME
Returns the current volume setting for the connector specified in *ulValue* of MCI_STATUS_PARMS as a percentage of the maximum achievable effect. MCI_STATUS_CONNECTOR must be specified.

CD Audio Extensions

The following additional status items apply to CD audio devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_CD_STATUS_TRACK_TYPE
Returns one of the following:

- MCI_CD_TRACK_AUDIO
- MCI_CD_TRACK_DATA
- MCI_CD_TRACK_OTHER

MCI_CD_STATUS_TRACK_COPYPERMITTED
Returns MCI_TRUE if digital copying is permitted; otherwise, returns MCI_FALSE.

MCI_CD_STATUS_TRACK_CHANNELS
Returns the number of audio channels on the track.

MCI_CD_STATUS_TRACK_PREEMPHASIS
Returns MCI_TRUE if the track was recorded with pre-emphasis; otherwise, returns MCI_FALSE.

Note: When used with the MCI_TRACK flag, these items return the status information of the specified track instead of the current track.

CD/XA Extensions

The following extensions apply to CD-XA devices and can be specified for the *ulItem* field of the data structure pointed to by *pParam2*:

MCI_CDXA_STATUS_CHANNEL
Returns the destination of the data in channel *ulChannel*. Returns one of the following:

- MCI_CDXA_AUDIO_DEVICE
- MCI_CDXA_AUDIO_BUFFER
- MCI_CDXA_VIDEO_BUFFER
- MCI_CDXA_DATA_BUFFER
- MCI_CDXA_NONE

Digital Video Extensions

The following additional status items apply to digital video devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag.

MCI_DGV_STATUS_FORMATTAG
Returns WAVE_FORMAT_PCM, the only format currently supported by the digital video device. If a movie is loaded that contains a format other than PCM, the format used in the movie will be returned.

MCI_DGV_STATUS_DROPPED_FRAME_PCT
Returns the percentage of dropped frames for recording or playback operations. The value returned is in the range 0-100, where a value of zero indicates that no frame drops are occurring or have occurred and a value of 100 indicates that all frames are being dropped or have been dropped. This status value can be queried during a recording operation to obtain the cumulative percentage of frame drops that have occurred since recording began, or during playback to obtain the cumulative percentage of frame drops that have occurred since playback began or was resumed after a seek, pause, or stop. If the value is queried when the device is stopped, the percentage of dropped frames accumulated at the end of the last playback or recording operation that was performed is returned. A value of zero is returned if no playback or recording operations have been performed, the device is seeking or has been seeked, the device is

paused or stopped, or the device is playing in scan mode.

MCI_DGV_STATUS_SAMPLESPERSEC
Returns the currently set samples per second used for playing, recording, and saving.

MCI_DGV_STATUS_BITSPERSAMPLE
Returns the currently set bits per sample used for playing, recording, and saving.

MCI_DGV_STATUS_CHANNELS
Returns the currently set number of channels used for playing, recording, and saving.

MCI_DGV_STATUS_HWND
Returns the handle of the playback window.

MCI_DGV_STATUS_VIDEO_COMPRESSION
Returns the current FOURCC compression format for recording of motion video. Only symmetric compressors will be enabled for real-time recording.

MCI_DGV_STATUS_VIDEO_QUALITY
Returns the currently set compression quality level for recording of motion video.

MCI_DGV_STATUS_MONITOR
Returns MCI_ON or MCI_OFF to indicate whether monitoring of the incoming video signal is on or off.

MCI_DGV_STATUS_HWND_MONITOR
Returns the monitor window handle.

MCI_DGV_STATUS_REF_INTERVAL
Returns the value of *n* where *n* refers to a reference frame being inserted every *n*th frame.

MCI_DGV_STATUS_IMAGE_BITSPERPEL
Returns the pel format used for saving bitmaps.

MCI_DGV_STATUS_IMAGE_PELFORMAT
Returns the data format used of image data for the capture device. Possible values are:

- MMIO_RGB_5_6_5

  Each pixel is represented by 16 bits of data as follows:

15:5	Red level in the range 0-31
10:6	Green level in the range 0-63
4:5	Blue level in the range 0-31

- MMIO_YUV_4_1_1

  This format uses 16 bits per pixel, but uses 4-pixel horizontal chrominance subsampling. Each pixel has a unique luminance value (Y) with a single chrominance value (U and V) shared by four pixels. Y, U, and V all have 7 bits of significance in this format.

23:8	Red level in the range 0-255
15:8	Green level in the range 0-255
7:8	Blue level in the range 0-255

- MMIO_YUV_4_2_2

  4 bytes of Y, 2 bytes of U, 2 bytes of V; all 8-bit values in this form YUYVYUYV

MCI_DGV_STATUS_FORWARD
Returns MCI_TRUE if playing forward; otherwise returns MCI_FALSE.

MCI_DGV_STATUS_NORMAL_RATE
Returns the normal-play rate of the currently loaded motion video device element, in the current speed format, either as a percentage or in frames per second.

MCI_DGV_STATUS_VIDEO_X_EXTENT
Returns the horizontal (X) extent of the digital motion video image for the currently loaded motion video device element.

MCI_DGV_STATUS_VIDEO_Y_EXTENT
Returns the vertical (Y) extent of the digital motion video image for the currently loaded motion

video device element.

MCI_DGV_STATUS_BRIGHTNESS
Returns the brightness level.

MCI_DGV_STATUS_CONTRAST
Returns the contrast level.

MCI_DGV_STATUS_HUE
Returns the hue level.

MCI_DGV_STATUS_SATURATION
Returns the saturation level.

MCI_DGV_STATUS_RECORD_AUDIO
Returns MCI_ON or MCI_OFF to indicate whether recording the audio soundtrack has been turned on or off.

MCI_DGV_STATUS_SPEED
Returns the digital video speed in frames per second.

MCI_DGV_STATUS_TRANSPARENT_COLOR
Returns a value representing the transparent color used as the chroma-key on video overlay hardware.

MCI_DGV_STATUS_VIDEO_RECORD_FRAME_DURATION
Returns the frame rate for recording as the time duration of each frame in microseconds.

MCI_DGV_STATUS_TUNER_TV_CHANNEL
This flag returns the channel that the tuner device is tuned to.

MCI_DGV_STATUS_TUNER_HIGH_TV_CHANNEL
This flag returns the highest channel for the region.

MCI_DGV_STATUS_TUNER_LOW_TV_CHANNEL
This flag returns the lowest channel for the region.

MCI_DGV_STATUS_TUNER_FINETUNE
This flag returns the fine-tuning value that the tuner device is tuned to.

MCI_DGV_STATUS_TUNER_FREQUENCY
This flag returns the frequency value that the tuner device is tuned to.

MCI_DGV_STATUS_VALID_SIGNAL
This flag returns MCI_TRUE if there is a signal present.

Sequencer Extensions

The following additional status items apply to MIDI sequencer devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_SEQ_STATUS_DIVTYPE
Returns one of the following values as the current division type of a sequence:

- MCI_SEQ_DIV_PPQN
- MCI_SEQ_DIV_SMPTE_24
- MCI_SEQ_DIV_SMPTE_25
- MCI_SEQ_DIV_SMPTE_25
- MCI_SEQ_DIV_SMPTE_30
- MCI_SEQ_DIV_SMPTE_30DROP

MCI_SEQ_STATUS_MASTER
Returns the synchronization type used for master operation.

MCI_SEQ_STATUS_OFFSET
Returns the current SMPTE offset of a sequence.

MCI_SEQ_STATUS_PORT
Returns the MIDI device ID for the current port used by the sequence.

MCI_SEQ_STATUS_SLAVE
Returns the synchronization type used for slave operation.

MCI_SEQ_STATUS_TEMPO
Returns the current tempo of a MIDI sequence in beats-per-minute for PPQN files, or frames-per-second for SMPTE files. Currently this function is not supported by the IBM sequencer.

Videodisc Extensions

The following additional status items apply to videodisc devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_VD_STATUS_SPEED
Returns the speed in the currently set speed format.

MCI_VD_STATUS_FORWARD
Returns MCI_TRUE if playing forward; otherwise, returns MCI_FALSE.

MCI_VD_MEDIA_TYPE
Returns one of the following:

- MCI_VD_MEDIA_CAV
- MCI_VD_MEDIA_CLV
- MCI_VD_MEDIA_OTHER

MCI_VD_STATUS_SIDE
Returns 1 or 2 to indicate which side of the disc is loaded.

MCI_VD_STATUS_DISC_SIZE
Returns the size of the loaded disc in inches (8 or 12).

Video Overlay Extensions

The following additional items apply to video overlay devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag.

MCI_OVLY_STATUS_HWND
Returns the handle of the playback window.

MCI_OVLY_STATUS_IMAGE_COMPRESSION
Returns the compression format of the currently loaded bitmap/image.

MCI_OVLY_STATUS_BITSPERPEL
Returns the number of bits per pel of the currently loaded bitmap/image. Return values include:

- MCI_IMG_PALETTE
- MCI_IMG_RGB
- MCI_IMG_YUV

MCI_OVLY_STATUS_PELFORMAT
Returns the pel format of the currently loaded bitmap/image.

MCI_OVLY_STATUS_BRIGHTNESS
Returns the brightness level.

MCI_OVLY_STATUS_CONTRAST
Returns the contrast level.

MCI_OVLY_STATUS_HUE
Returns the hue level.

MCI_OVLY_STATUS_SATURATION
Returns the saturation level.

MCI_OVLY_STATUS_SHARPNESS
Returns the sharpness level.

MCI_OVLY_STATUS_TRANSPARENT_COLOR
Returns a value representing the RGB value or palette value, which specifies the transparent color. RGB values are returned as a 32-bit RGB2 data item.

MCI_OVLY_STATUS_TRANSPARENT_TYPE
Returns a value representing information to assist in interpreting the MCI_OVLY_STATUS_TRANSPARENT_COLOR.

Return values include:

- MCI_IMG_PALETTE
- MCI_IMG_RGB
- MCI_IMG_YUV

MCI_OVLY_STATUS_GREYSCALE
Returns MCI_ON or MCI_OFF.

MCI_OVLY_STATUS_IMAGE_COMPRESSION
Returns the compression type for saving still images.

MCI_OVLY_STATUS_IMAGE_BITSPERPEL
Returns the number of bits per pel used for the image file to be saved.

MCI_OVLY_STATUS_IMAGE_PELFORMAT
Returns the pel format used for saving bitmaps.

MCI_OVLY_STATUS_IMAGE_QUALITY
Returns the quality of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_X_EXTENT
Returns the width, in pels, of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_Y_EXTENT
Returns the height, in pels, of the image in the element buffer.

MCI_OVLY_STATUS_IMAGE_FILE_FORMAT
Returns the format in which an image capture will be stored when saved.

Wave Audio Extensions

The following additional status items apply to wave audio devices and can be specified for the *ulItem* field (of the data structure pointed to by *pParam2*) for use with the MCI_STATUS_ITEM flag:

MCI_WAVE_STATUS_FORMATTAG
Returns the currently set format tag used for playing, recording, and saving.

MCI_WAVE_STATUS_CHANNELS
Returns the currently set channel count used for playing, recording, and saving.

MCI_WAVE_STATUS_SAMPLESPERSEC
Returns the currently set samples per second used for playing, recording, and saving.

MCI_WAVE_STATUS_AVGBYTESPERSEC
Returns the currently set bytes per second used for playing, recording, and saving. Playback software can use this number to estimate required buffer sizes. Refer to the RIFF WAVE format documentation for more information.

MCI_WAVE_STATUS_BLOCKALIGN
Returns the currently set block alignment used for playing, recording, and saving.

MCI_WAVE_STATUS_BITSPERSAMPLE
Returns the currently set bits per sample used for playing, recording, and saving.

MCI_WAVE_STATUS_LEVEL
Returns the current record or playback level. The value is returned as an 8-bit or 16-bit value, depending on the sample size being used. The right or Mono channel level is returned in the low-order word. The left channel level is returned in the high-order word.

pParam2 (PMCI_STATUS_PARMS)
A pointer to the MCI_STATUS_PARMS data structure. Devices with extended command sets might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_CDXA_STATUS_PARMS
A pointer to the MCI_CDXA_STATUS_PARMS data structure.

rc (ULONG)

Note: The format of the *ulReturn* value in this structure is defined by the high-order word of the value returned by mciSendCommand. This value is used by mciSendString to determine how to convert the *ulReturn* value to string form. For a list of the possible format values, see the MMDRVOS2.H header file. If the low-order word returned is MCIERR_SUCCESS, the high-order word could be other errors or a value. A returned value defines the format of *ulReturn* as defined in MMDRVOS2.H. For example, 0x5000 = MCI_TRUE_FALSE_RETURN.

Return codes indicating success or type of failure:

MCIERR_SUCCESS	MMPM/2 command completed successfully.
MCIERR_OUT_OF_MEMORY	System out of memory.
MCIERR_INVALID_DEVICE_ID	Invalid device ID given.
MCIERR_MISSING_PARAMETER	Missing parameter for this command.
MCIERR_DRIVER	Internal MMPM/2 driver error.
MCIERR_INVALID_FLAG	Invalid flag specified for this command.
MCIERR_UNSUPPORTED_FLAG	Flag not supported by this MMPM/2 driver for this command.
MCIERR_MISSING_FLAG	Flag missing for this MMPM/2 command.
MCIERR_UNSUPPORTED_FUNCTION	Function not supported by the media driver being used.
MCIERR_INVALID_ITEM_FLAG	Invalid item flag specified for this command.
MCIERR_TUNER_NO_HW	Device has no tuner support.
MCIERR_TUNER_MODE	Frequency was last set directly. MCI_DGV_STATUS_TUNER_TV_CHANNEL and MCI_DGV_STATUS_TUNER_FINETUNE cannot be used. Use MCI_DGV_STATUS_FREQUENCY.
MCIERR_SIGNAL_INVALID	No valid signal present.

MCI_STATUS - Remarks

The parameters and flags for this message vary according to the selected device. All devices support this message and the applicable status items for each device are listed with each parameter. See the MCI_SET message for the values which can be returned for each particular item.

If the frequency was set on the MCI_SETTUNER command using the MCI_DGV_FREQUENCY flag then status of channel, region, and fine-tuning will return an MCIERR_TUNER_MODE error.

MCI_STATUS - Related Messages

- MCI_SET

MCI_STATUS - Example Code

The following code illustrates how to obtain information about the status of a media device.

```
USHORT   usDeviceID;
ULONG    ulError;
BOOL     disc_loaded;
                 /* Set to TRUE by this example if media is present */
MCI_STATUS_PARMS  mstatusp;

mstatusp.ulItem = MCI_STATUS_MEDIA_PRESENT;

ulError = mciSendCommand(usDeviceID,      /* Device ID              */
 MCI_STATUS,                              /* MCI status message     */
 MCI_WAIT | MCI_STATUS_ITEM,
                                          /* Flags for this message */
 (PVOID) &mstatusp,                       /* Data structure         */
 0);                                      /* No user parm           */

if (LOUSHORT(ulError) == MCIERR_SUCCESS)
  {
    disc_loaded = (BOOL) mstatusp.ulReturn;    /* Media present
                                                  status            */
  }
```

MCI_STATUS - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_STEP

MCI_STEP Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_STEP_FRAMES
This flag is used to set a step frames parameter. This provides step forward support. The step increment is specified in the *ulStep* field of the MCI_STEP_PARMS data structure.

MCI_STEP_REVERSE
This flag is used to set a steps in reverse parameter.

MCI_STEP Parameter - pParam2

pParam2 (PMCI_STEP_PARMS)
A pointer to the MCI_STEP_PARMS data structure.

MCI_STEP Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_ITEM_FLAG
Invalid status item flag given.

MCIERR_MISSING_ITEM
Missing status item flag.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_STEP - Description

This message is sent to step the player and is intended for videodisc players.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_STEP_FRAMES
This flag is used to set a step frames parameter. This provides step forward support. The step increment is specified in the *ulStep* field of the MCI_STEP_PARMS data structure.

MCI_STEP_REVERSE
This flag is used to set a steps in reverse parameter.

pParam2 (PMCI_STEP_PARMS)
A pointer to the MCI_STEP_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag is invalid (*ulParam1*).

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_ITEM_FLAG
Invalid status item flag given.

MCIERR_MISSING_ITEM
Missing status item flag.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_STEP - Remarks

The step can be sent for either forward-frame or reverse-frame operation.

If you are using an application-defined window and your application is running on a system without direct-access device driver support for motion video, do *not* issue MCI_STEP with the MCI_WAIT flag specified unless the thread issuing the message is separate from the thread reading the message queue.

MCI_STEP - Default Processing

If no flags are specified, MCI_STEP steps one frame forward. If only MCI_STEP_REVERSE flag is specified, MCI_STEP steps one frame backward.

MCI_STEP - Related Messages

- MCI_PLAY
- MCI_PAUSE
- MCI_RECORD
- MCI_RESUME

MCI_STEP - Example Code

The following code illustrates how to a step a player 10 frames.

```
USHORT          usDeviceID;
MCI_STEP_PARMS   mstepp;

                                          /* Step the device 10 frames */

                    /* Assumes time format for device set to frames */
mstepp.ulStep = (ULONG) 10;

mciSendCommand( usDeviceID,             /* Device ID                 */
                MCI_STEP,               /* MCI step message          */
                MCI_WAIT | MCI_STEP_FRAMES,
                                        /* Flags for this message    */
                (PVOID) &mstepp,        /* Data structure            */
                0);                     /* No user parm              */
```

MCI_STEP - Topics

Select an item:

MCI_STOP

MCI_STOP Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_STOP Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_STOP Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID

The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
: The device ID is currently inactive. Issue MCI_ACQUIREDEVICE MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
: A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_HARDWARE
: Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
: Unsupported function.

MCIERR_INVALID_FLAG
: Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCIERR_INVALID_ITEM_FLAG
: Invalid status item flag given.

MCIERR_MISSING_ITEM
: Missing status item flag.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCI_STOP - Description

This message is sent to stop playback or recording.

If MCI_STOP is issued, video recording is stopped regardless of whether a TO position is reached. Once video recording is stopped, it cannot be restarted without overwriting what was previously recorded.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
: A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INSTANCE_INACTIVE
The device ID is currently inactive. Issue MCI_ACQUIREDEVICE MCI_ACQUIREDEVICE to make device ID active.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_HARDWARE
Device hardware error.

MCIERR_UNSUPPORTED_FUNCTION
Unsupported function.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_INVALID_ITEM_FLAG
Invalid status item flag given.

MCIERR_MISSING_ITEM
Missing status item flag.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCI_STOP - Remarks

If playback or recording is to be restarted with minimal latency, MCI_PAUSE should be used.

MCI_STOP - Related Messages

- MCI_PLAY
- MCI_RECORD

MCI_STOP - Example Code

The following code illustrates how to stop an audio or video device during playback or recording, and receive notification upon completion.

```
USHORT             usDeviceID;
HWND               hwndMyWindow;
MCI_GENERIC_PARMS mciGenericParms;        /* Info data structure
```

```
                                                 for command            */

                 /* Assign hwndCallback the handle to the PM Window */
mciGenericParms.hwndCallback = hwndMyWindow;

                                              /* Stop the device        */

mciSendCommand( usDeviceID,                   /* Device ID              */
 MCI_STOP,                                    /* MCI stop message       */
 MCI_NOTIFY,                                  /* Flag for this message  */
 (PVOID) &mciGenericParms,                    /* Data structure         */
 0);                                          /* No user parm           */
```

MCI_STOP - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_SYSINFO

MCI_SYSINFO Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags. If the size of the buffer passed in is too small to hold all the data returned, then the *ulRetSize* field of the MCI_SYSINFO_PARMS data structure will contain the required buffer size, MCIERR_INVALID_BUFFER will be returned, and the buffer will contain only as much of the SYSINFO data as its size permits. Only one MCI_SYSINFO_*xxxxx* flag can be used per MCI_SYSINFO message. The MCI_SYSINFO_NAME and MCI_SYSINFO_QUANTITY flags are mutually exclusive.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SYSINFO_INSTALLNAME
This flag returns the name used to install the device.

MCI_SYSINFO_QUANTITY
This flag sets the media to return the number of devices of the given type. If the MCI_SYSINFO_OPEN flag is set, the number of open devices is returned.

MCI_SYSINFO_NAME
This flag is used to select a number or device ordinal parameter. The media returns the name(s) of a device that satisfies the query. If more than one name is returned then the names are separated by a single blank and the string is null terminated.

MCI_SYSINFO_OPEN
This flag returns the number or name of open devices.

MCI_SYSINFO_ITEM

This flag indicates that the *ulItem* field contains a constant that indicates the desired MCI_SYSINFO action as indicated by one of the following values:

MCI_SYSINFO_INSTALL_DRIVER

This message creates or updates a logical device entry in the INI file. The *pSysInfoParm* field points to the MCI_SYSINFO_LOGDEVICE data structure. The driver becomes active the next time the system is started.

MCI_SYSINFO_QUERY_DRIVER

This message queries the information for the driver indicated in the *szInstallName* field of the MCI_SYSINFO_LOGDEVICE data structure. The *pSysInfoParm* field points to the MCI_SYSINFO_LOGDEVICE data structure.

MCI_SYSINFO_INI_LOCK

Writes out and then locks the MMPM2.INI file from updates.

MCI_SYSINFO_DELETE_DRIVER

This message removes the specified driver from the INI file. The *pSysInfoParm* field points to the installation name.

MCI_SYSINFO_SET_PARAMS

This message sets the device-specific parameters for a particular device. Device-specific parameters should be printable ASCII characters only so that response files can be supported. The *pSysInfoParm* field points to the MCI_SYSINFO_DEVPARAMS data structure.

MCI_SYSINFO_QUERY_PARAMS

This message retrieves the device-specific parameters for a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_DEVPARAMS data structure.

MCI_SYSINFO_SET_CONNECTORS

This message sets the logical connector information for a particular device. The connector array defined by *ConnectorList* (in the MCI_SYSINFO_CONPARAMS data structure) is a list of the connectors in sequential order. For example, ConnectorList[0] is connector index 1. The *pSysInfoParm* field points to the MCI_SYSINFO_CONPARAMS data structure.

MCI_SYSINFO_QUERY_CONNECTORS

This message retrieves the device connector information for a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_CONPARAMS data structure.

MCI_SYSINFO_SET_EXTENSIONS

This message sets the file extension associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_EXTENSION data structure. Extensions are unique across installation names. That is, no two installation names can have the same extension.

MCI_SYSINFO_QUERY_EXTENSIONS

This message queries the file extensions associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_EXTENSION data structure.

MCI_SYSINFO_SET_TYPES

This message sets the extended type attribute associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_TYPES data structure.

MCI_SYSINFO_QUERY_TYPES

This message queries query the extended type attributes associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_TYPES data structure.

MCI_SYSINFO_SET_ALIAS

This message associates an alias to a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_ALIAS data structure.

MCI_SYSINFO_QUERY_NAMES

This message queries the names associated with a particular device. This message will accept any of the three types of names or device type and device ordinal and fill in the remaining structure if possible. If the device type is given and 0 for the device ordinal then the first device of that type is returned. Only one non-null name or 0 in device type field on input is allowed. The *pSysInfoParm* field points to the MCI_SYSINFO_QUERY_NAME data structure.

MCI_SYSINFO_SET_DEFAULT

This message sets a device as the default for its device type. If another device is already the default for this device type, then it will be superseded by the new device. The *pSysInfoParm* field points to the MCI_SYSINFO_DEFAULTDEVICE data structure.

MCI_SYSINFO_QUERY_DEFAULT
: This message queries the default device for a given device type. If no explicit default exists, then the first device of the indicated type is implicitly the default. The *pSysInfoParm* field points to the MCI_SYSINFO_DEFAULTDEVICE data structure.

MCI_SYSINFO Parameter - pParam2

pParam2 (PMCI_SYSINFO_PARMS)
A pointer to the MCI_SYSINFO_PARMS structure.

MCI_SYSINFO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
If the function succeeds, 0 is returned.

MCIERR_MISSING_FLAG
A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags cannot be used together.

MCIERR_MISSING_PARAMETER
Required parameter is missing.

MCIERR_INVALID_BUFFER
Invalid return buffer given.

MCIERR_DUPLICATE_ALIAS
Alias already exists.

MCIERR_DUPLICATE_EXTENSION
Extension already exists.

MCIERR_NODEFAULT_DEVICE
No device of this type exists.

MCIERR_DEVICE_NOT_FOUND
Device not found for this query.

MCIERR_DUPLICATE_EA
The given EA already exists for another device.

MCI_SYSINFO - Description

This message returns information about media control devices and device instances.

ulParam1 (ULONG)

This parameter can contain any of the following flags. If the size of the buffer passed in is too small to hold all the data returned, then the *ulRetSize* field of the MCI_SYSINFO_PARMS data structure will contain the required buffer size, MCIERR_INVALID_BUFFER will be returned, and the buffer will contain only as much of the SYSINFO data as its size permits. Only one MCI_SYSINFO_*xxxxx* flag can be used per MCI_SYSINFO message. The MCI_SYSINFO_NAME and MCI_SYSINFO_QUANTITY flags are mutually exclusive.

MCI_WAIT
: Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_SYSINFO_INSTALLNAME
: This flag returns the name used to install the device.

MCI_SYSINFO_QUANTITY
: This flag sets the media to return the number of devices of the given type. If the MCI_SYSINFO_OPEN flag is set, the number of open devices is returned.

MCI_SYSINFO_NAME
: This flag is used to select a number or device ordinal parameter. The media returns the name(s) of a device that satisfies the query. If more than one name is returned then the names are separated by a single blank and the string is null terminated.

MCI_SYSINFO_OPEN
: This flag returns the number or name of open devices.

MCI_SYSINFO_ITEM
: This flag indicates that the *ulItem* field contains a constant that indicates the desired MCI_SYSINFO action as indicated by one of the following values:

  MCI_SYSINFO_INSTALL_DRIVER
  : This message creates or updates a logical device entry in the INI file. The *pSysInfoParm* field points to the MCI_SYSINFO_LOGDEVICE data structure. The driver becomes active the next time the system is started.

  MCI_SYSINFO_QUERY_DRIVER
  : This message queries the information for the driver indicated in the *szInstallName* field of the MCI_SYSINFO_LOGDEVICE data structure. The *pSysInfoParm* field points to the MCI_SYSINFO_LOGDEVICE data structure.

  MCI_SYSINFO_INI_LOCK
  : Writes out and then locks the MMPM2.INI file from updates.

  MCI_SYSINFO_DELETE_DRIVER
  : This message removes the specified driver from the INI file. The *pSysInfoParm* field points to the installation name.

  MCI_SYSINFO_SET_PARAMS
  : This message sets the device-specific parameters for a particular device. Device-specific parameters should be printable ASCII characters only so that response files can be supported. The *pSysInfoParm* field points to the MCI_SYSINFO_DEVPARAMS data structure.

  MCI_SYSINFO_QUERY_PARAMS
  : This message retrieves the device-specific parameters for a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_DEVPARAMS data structure.

  MCI_SYSINFO_SET_CONNECTORS
  : This message sets the logical connector information for a particular device. The connector array defined by *ConnectorList* (in the MCI_SYSINFO_CONPARAMS data structure) is a list of the connectors in sequential order. For example, ConnectorList[0] is connector index 1. The *pSysInfoParm* field points to the MCI_SYSINFO_CONPARAMS data structure.

MCI_SYSINFO_QUERY_CONNECTORS
: This message retrieves the device connector information for a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_CONPARAMS data structure.

MCI_SYSINFO_SET_EXTENSIONS
: This message sets the file extension associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_EXTENSION data structure. Extensions are unique across installation names. That is, no two installation names can have the same extension.

MCI_SYSINFO_QUERY_EXTENSIONS
: This message queries the file extensions associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_EXTENSION data structure.

MCI_SYSINFO_SET_TYPES
: This message sets the extended type attribute associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_TYPES data structure.

MCI_SYSINFO_QUERY_TYPES
: This message queries query the extended type attributes associated with a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_TYPES data structure.

MCI_SYSINFO_SET_ALIAS
: This message associates an alias to a particular device. The *pSysInfoParm* field points to the MCI_SYSINFO_ALIAS data structure.

MCI_SYSINFO_QUERY_NAMES
: This message queries the names associated with a particular device. This message will accept any of the three types of names or device type and device ordinal and fill in the remaining structure if possible. If the device type is given and 0 for the device ordinal then the first device of that type is returned. Only one non-null name or 0 in device type field on input is allowed. The *pSysInfoParm* field points to the MCI_SYSINFO_QUERY_NAME data structure.

MCI_SYSINFO_SET_DEFAULT
: This message sets a device as the default for its device type. If another device is already the default for this device type, then it will be superseded by the new device. The *pSysInfoParm* field points to the MCI_SYSINFO_DEFAULTDEVICE data structure.

MCI_SYSINFO_QUERY_DEFAULT
: This message queries the default device for a given device type. If no explicit default exists, then the first device of the indicated type is implicitly the default. The *pSysInfoParm* field points to the MCI_SYSINFO_DEFAULTDEVICE data structure.

pParam2 (PMCI_SYSINFO_PARMS)
: A pointer to the MCI_SYSINFO_PARMS structure.

rc (ULONG)
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
: If the function succeeds, 0 is returned.

MCIERR_MISSING_FLAG
: A required flag is missing.

MCIERR_UNSUPPORTED_FLAG
: Given flag is unsupported for this device.

MCIERR_INVALID_CALLBACK_HANDLE
: Given callback handle is invalid.

MCIERR_INVALID_FLAG
: Flag (*ulParam1*) is invalid.

MCIERR_FLAGS_NOT_COMPATIBLE
: Flags cannot be used together.

MCIERR_MISSING_PARAMETER
: Required parameter is missing.

MCIERR_INVALID_BUFFER
: Invalid return buffer given.

MCIERR_DUPLICATE_ALIAS

Alias already exists.

MCIERR_DUPLICATE_EXTENSION
Extension already exists.

MCIERR_NODEFAULT_DEVICE
No device of this type exists.

MCIERR_DEVICE_NOT_FOUND
Device not found for this query.

MCIERR_DUPLICATE_EA
The given EA already exists for another device.

MCI_SYSINFO - Remarks

The *usDeviceType* field of the MCI_SYSINFO_PARMS data structure is used to indicate the device type of the query. Specifying MCI_ALL_DEVICE_ID as the *usDeviceType* parameter, the media control interface returns information on all devices open by the current process. If MCI_ALL_DEVICE_ID and MCI_SYSINFO_NAME are specified together then the *ulNumber* field of the MCI_SYSINFO_PARMS data structure is ignored and names for all devices are returned. The names will be returned separated by a single blank and null terminated.

The MCI_SYSINFO *ulItem* actions are intended to be used by applications that need to update the MMPM2.INI file, such as installation and setup. The MCI_SYSINFO *ulItem* actions MCI_SYSINFO_INSTALL_DRIVER and MCI_SYSINFO_DELETE_DRIVER do not take effect until the next time the system is started. All other MCI_SYSINFO *ulItem* actions take effect during the current session.

MCI_SYSINFO - Default Processing

If MCI_SYSINFO_QUERY_DEFAULT is specified and no explicit default device type exists, then the first device of the indicated type is implicitly the default.

MCI_SYSINFO - Example Code

The following code illustrates how to determine the number of waveform devices installed.

```
#define  RETBUFSIZE 128

MCI_SYSINFO_PARMS  SysInfo;
CHAR  SysInfoRet[RETBUFSIZE];
                                    /*  Set unused fields to zero.  */
memset(&SysInfo, 0x00, sizeof(MCI_SYSINFO_PARMS));
SysInfo.usDeviceType  = MCI_DEVTYPE_WAVEFORM_AUDIO;
                                    /* Device type                  */
SysInfo.pszReturn = (PSZ) &SysInfoRet;
                                    /* Pointer to return buffer     */
SysInfo.ulRetSize = RETBUFSIZE;

        /* Determine the number of waveform audio devices installed */

mciSendCommand (0,                  /* Don't know device ID yet     */
 MCI_SYSINFO,                       /* MCI sysinfo message          */
 MCI_SYSINFO_QUANTITY | MCI_WAIT,
                                    /* Flags for this message       */
 (PVOID)&SysInfo,                   /* Data structure               */
 0);                                /* No user parm                 */

        /* SysInfoRet now contains number of wave audio devices.    */
```

MCI_SYSINFO - Topics

Select an item:

MCI_UNDO

MCI_UNDO Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

MCI_UNDO Parameter - pParam2

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

MCI_UNDO Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS

The UNDO was successful.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_CANNOT_UNDO
Undo is not possible in the current state.

MCI_UNDO - Description

This message undoes the operation most recently performed by cut, paste, or delete.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

pParam2 (PMCI_GENERIC_PARMS)
A pointer to the default media control interface parameter data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
The UNDO was successful.

MCIERR_INVALID_DEVICE_ID
The device ID is not valid.

MCIERR_INVALID_FLAG
Flag (*ulParam1*) is invalid.

MCIERR_INSTANCE_INACTIVE
The device is currently inactive. Issue MCI_ACQUIREDEVICE to make the device context active.

MCIERR_INVALID_CALLBACK_HANDLE
Given callback handle is invalid.

MCIERR_CANNOT_UNDO
Undo is not possible in the current state.

MCI_UNDO - Remarks

After an undo operation, the media position is at the beginning of the media.

Undo is unlimited. However, after a save, any previous editing actions (such as cut, delete, paste) are cleared and cannot be undone. If there are no possible actions to be undone (the file is in the state where the last change was made) then MCIERR_CANNOT_UNDO is returned.

If undo interrupts an in-progress operation, such as play, the command is aborted and an MM_MCINOTIFY message is sent to the application.

Not all devices support this command. Use the MCI_GETDEVCAPS message to determine whether the device supports MCI_UNDO.

MCI_UNDO - Related Messages

- MCI_COPY
- MCI_CUT
- MCI_PASTE
- MCI_DELETE
- MCI_REDO

MCI_UNDO - Example Code

The following code illustrates undoing the last operation.

```
USHORT                 usDeviceID;
MCI_EDIT_PARMS         mep;

mep.hwndCallback = hwndMyWindow;

mciSendCommand( usDeviceID,
                MCI_UNDO,
                MCI_NOTIFY,
                &mep,
                0 );
```

MCI_UNDO - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_UNFREEZE

MCI_UNFREEZE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Video Overlay Extensions

MCI_OVLY_FREEZE_RECT
The *rect* field of the data structure identified by *pParam2* contains a valid rectangle. If the MCI_OVLY_FREEZE_RECT parameter is not specified, the entire video destination rectangle is unfrozen.

MCI_OVLY_FREEZE_RECT_OUTSIDE
Indicates the area outside the specified rectangle is to be unfrozen.

MCI_UNFREEZE Parameter - pParam2

pParam2 (PMCI_OVLY_RECT_PARMS)
A pointer to the MCI_OVLY_RECT_PARMS data structure.

MCI_UNFREEZE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG

Invalid flag specified for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available. (For example, video has been set off.)

MCI_UNFREEZE - Description

This message restores motion to an area of the display frozen with MCI_FREEZE or MCI_RESTORE.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Video Overlay Extensions

MCI_OVLY_FREEZE_RECT
The *rect* field of the data structure identified by *pParam2* contains a valid rectangle. If the MCI_OVLY_FREEZE_RECT parameter is not specified, the entire video destination rectangle is unfrozen.

MCI_OVLY_FREEZE_RECT_OUTSIDE
Indicates the area outside the specified rectangle is to be unfrozen.

pParam2 (PMCI_OVLY_RECT_PARMS)
A pointer to the MCI_OVLY_RECT_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCIERR_OVLY_INVALID_RECT
An invalid rectangle parameter was specified.

MCIERR_OVLY_NOT_AVAILABLE
The requested action is not available. (For example, video has been set off.)

MCI_UNFREEZE - Remarks

Areas outside the current video destination region will be unaffected. Multiple MCI_FREEZE and MCI_UNFREEZE messages can be issued sequentially to build up a complex region of frozen and unfrozen video.

MCI_UNFREEZE - Example Code

The following code illustrates how to restore motion to an area of the display frozen with MCI_FREEZE.

```
MCI_VID_RECT_PARMS mciUnFreezeParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;

/* An example of unfreezing a sub-rectangle */
memset (&mciUnFreezeParms, 0x00, sizeof (MCI_VID_RECT_PARMS));
mciUnFreezeParms.hwndCallback = hwndNotify;
mciUnFreezeParms.rc.xLeft   = lX1;
mciUnFreezeParms.rc.yBottom = lY1;
mciUnFreezeParms.rc.xRight  = lX2;
mciUnFreezeParms.rc.yTop    = lY2;


ulReturn = mciSendCommand(usDeviceID, MCI_UNFREEZE,
               MCI_WAIT | MCI_OVLY_FREEZE_RECT,
               (PVOID)&mciUnFreezeParms,
              usUserParm);
```

MCI_UNFREEZE - Topics

Select an item:
Description
Returns
Remarks
Example Code
Glossary

MCI_WHERE

MCI_WHERE Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_WHERE_DESTINATION
This flag indicates that the destination display rectangle should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_DGV_WHERE_SOURCE
This flag indicates that the video source rectangle should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_DGV_WHERE_ADJUSTED
Used with either MCI_DGV_WHERE_SOURCE and MCI_DGV_RECORD or MCI_DGV_WHERE_DESTINATION and MCI_DGV_RECORD. When MCI_DGV_WHERE_ADJUSTED is specified, these commands return the coordinates that will actually be used to record a movie or get an image buffer based on what was set with MCI_PUT in combination with the capabilities of the capture hardware.

MCI_DGV_WHERE_WINDOW
This flag indicates the current location of the video window relative to its parent should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_DGV_MONITOR
This flag indicates the window size and position for the monitor window.

MCI_DGV_RECORD
This flag indicates the source and destination rectangles for the video capture.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_WHERE_DESTINATION
This flag indicates that the destination display rectangle should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_OVLY_WHERE_SOURCE
This flag indicates that the video overlay source rectangle should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_OVLY_WHERE_WINDOW
This flag indicates the current location of the video window relative to its parent should be returned in the *rc* field of the data structure identified by *pParam2*.

MCI_WHERE Parameter - pParam2

pParam2 (PMCI_VID_RECT_PARMS)
A pointer to the MCI_VID_RECT_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer

to a device-specific data structure as follows:

PMCI_DGV_RECT_PARMS
A pointer to the MCI_DGV_RECT_PARMS data structure.

PMCI_OVLY_RECT_PARMS
A pointer to the MCI_OVLY_RECT_PARMS data structure.

MCI_WHERE Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_WHERE - Description

This message returns the source and destination rectangles, and the location of the video window.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_WHERE_DESTINATION
: This flag indicates that the destination display rectangle should be returned in the *rc* field of the data structure identified by *pParam2* .

MCI_DGV_WHERE_SOURCE
: This flag indicates that the video source rectangle should be returned in the *rc* field of the data structure identified by *pParam2* .

MCI_DGV_WHERE_ADJUSTED
: Used with either MCI_DGV_WHERE_SOURCE and MCI_DGV_RECORD or MCI_DGV_WHERE_DESTINATION and MCI_DGV_RECORD. When MCI_DGV_WHERE_ADJUSTED is specified, these commands return the coordinates that will actually be used to record a movie or get an image buffer based on what was set with MCI_PUT in combination with the capabilities of the capture hardware.

MCI_DGV_WHERE_WINDOW
: This flag indicates the current location of the video window relative to its parent should be returned in the *rc* field of the data structure identified by *pParam2* .

MCI_DGV_MONITOR
: This flag indicates the window size and position for the monitor window.

MCI_DGV_RECORD
: This flag indicates the source and destination rectangles for the video capture.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_WHERE_DESTINATION
: This flag indicates that the destination display rectangle should be returned in the *rc* field of the data structure identified by *pParam2* .

MCI_OVLY_WHERE_SOURCE
: This flag indicates that the video overlay source rectangle should be returned in the *rc* field of the data structure identified by *pParam2* .

MCI_OVLY_WHERE_WINDOW
: This flag indicates the current location of the video window relative to its parent should be returned in the *rc* field of the data structure identified by *pParam2* .

pParam2 (PMCI_VID_RECT_PARMS)
: A pointer to the MCI_VID_RECT_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_RECT_PARMS
: A pointer to the MCI_DGV_RECT_PARMS data structure.

PMCI_OVLY_RECT_PARMS
: A pointer to the MCI_OVLY_RECT_PARMS data structure.

rc (ULONG)
: Return codes indicating success or type of failure:

MCIERR_SUCCESS
: MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
: System out of memory.

MCIERR_INVALID_DEVICE_ID
: Invalid device ID given.

MCIERR_MISSING_PARAMETER
: Missing parameter for this command.

MCIERR_DRIVER
: Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG

Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_WHERE - Remarks

The parameters and flags vary according to the selected device.

Note: A pointer to the rectangle is returned in the *rc* field of the data structure identified by *pParam2* .

MCI_WHERE - Related Messages

- MCI_WINDOW

MCI_WHERE - Example Code

The following code illustrates how to return the video destination rectangle with MCI_WHERE.

```
MCI_DGV_RECT_PARMS  mciRectParms;
USHORT  usUserParm = 0;
ULONG   ulReturn;
CHAR    szText[255];
CHAR    szValue[20];
LONG    lX1, lX2, lY1, lY2;

/* A sample to query the current destination   */
/* video sub-rectangle within the video window */
memset (&mciRectParms, 0x00, sizeof (MCI_DGV_RECT_PARMS));
mciRectParms.hwndCallback = hwndNotify;

ulReturn = mciSendCommand(usDeviceID, MCI_WHERE,
               MCI_WAIT | MCI_DGV_WHERE_DESTINATION,
               (PVOID)&mciRectParms,
               usUserParm);

lX1 = mciRectParms.rc.xLeft;
lY1 = mciRectParms.rc.yBottom;
lX2 = mciRectParms.rc.xRight;
lY2 = mciRectParms.rc.yTop;
```

MCI_WHERE - Topics

Select an item:
Description
Returns
Remarks
Related Messages
Example Code
Glossary

MCI_WINDOW

MCI_WINDOW Parameter - ulParam1

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_MONITOR
This flag indicates functions associated with the MCI_WINDOW message are to be applied to the monitor window. The monitor window output can be directed to an application-specified window in the same manner as video playback.

MCI_DGV_WINDOW_HWND
This flag indicates the handle of the application window to be used for video is included in the *hwndDest* field of the data structure identified by *pParam2* .

MCI_DGV_WINDOW_DEFAULT
This flag indicates the default video window should be used as the target for video.

MCI_DGV_WINDOW_STATE
This flag indicates the *usCmdShow* field of the data structure identified by *pParam2* contains one of the following parameters for setting the window state:

- SWP_ACTIVATE
- SWP_DEACTIVATE
- SWP_HIDE
- SWP_MAXIMIZE
- SWP_MINIMIZE
- SWP_RESTORE
- SWP_SHOW

Note: The MCI_DGV_WINDOW_STATE flag only applies to the default window and will *not* affect an application-supplied alternate video window. Specifying MCI_DGV_WINDOW_DEFAULT in conjunction with the MCI_DGV_WINDOW_STATE flag will result in an error.

MCI_DGV_WINDOW_TEXT

This flag indicates the *pszText* field of the data structure identified by *pParam2* contains a pointer to a buffer containing the caption used for the window.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_WINDOW_DEFAULT
Indicates that the default video window should be used as the target for video.

MCI_OVLY_WINDOW_HWND
This flag indicates the handle of the application window to be used for video. It is included in the *hwndDest* field of the data structure identified by *pParam2* .

MCI_OVLY_WINDOW_STATE
This flag indicates the *usCmdShow* field of the data structure identified by *pParam2* contains a parameter for setting the window state. Window states include:

- SWP_ACTIVATE
- SWP_DEACTIVATE
- SWP_HIDE
- SWP_MAXIMIZE
- SWP_MINIMIZE
- SWP_RESTORE
- SWP_SHOW

Note: The MCI_OVLY_WINDOW_STATE flag only applies to the default window and will *not* affect an application-supplied alternate video window.

MCI_OVLY_WINDOW_TEXT
Indicates that the *pszText* field of the data structure identified by *pParam2* contains a pointer to a buffer containing the caption used for the window.

Note: The MCI_OVLY_WINDOW_TEXT flag only applies to the default window and will *not* affect an application-supplied alternate video window.

MCI_WINDOW Parameter - pParam2

pParam2 (PMCI_VID_WINDOW_PARMS)
A pointer to the MCI_VID_WINDOW_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_WINDOW_PARMS
A pointer to an MCI_DGV_WINDOW_PARMS data structure.

PMCI_OVLY_WINDOW_PARMS
A pointer to an MCI_OVLY_WINDOW_PARMS data structure.

MCI_WINDOW Return Value - rc

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG
Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_WINDOW - Description

This message specifies the window and the window characteristics that a graphic device should use for display.

ulParam1 (ULONG)
This parameter can contain any of the following flags:

MCI_NOTIFY
A notification message will be posted to the window specified in the *hwndCallback* parameter of the data structure pointed to by the *pParam2* parameter. The notification will be posted when the action indicated by this message is completed or when an error occurs.

MCI_WAIT
Control is not to be returned until the action indicated by this message is completed or an error occurs.

Digital Video Extensions

The following additional flags apply to digital video devices:

MCI_DGV_MONITOR
This flag indicates functions associated with the MCI_WINDOW message are to be applied to the monitor window. The monitor window output can be directed to an application-specified window in the same manner as video playback.

MCI_DGV_WINDOW_HWND
This flag indicates the handle of the application window to be used for video is included in the *hwndDest* field of the data structure identified by *pParam2*.

MCI_DGV_WINDOW_DEFAULT
This flag indicates the default video window should be used as the target for video.

MCI_DGV_WINDOW_STATE
This flag indicates the *usCmdShow* field of the data structure identified by *pParam2* contains one of the following parameters for setting the window state:

- SWP_ACTIVATE
- SWP_DEACTIVATE
- SWP_HIDE
- SWP_MAXIMIZE
- SWP_MINIMIZE

- SWP_RESTORE
- SWP_SHOW

Note: The MCI_DGV_WINDOW_STATE flag only applies to the default window and will *not* affect an application-supplied alternate video window. Specifying MCI_DGV_WINDOW_DEFAULT in conjunction with the MCI_DGV_WINDOW_STATE flag will result in an error.

MCI_DGV_WINDOW_TEXT
This flag indicates the *pszText* field of the data structure identified by *pParam2* contains a pointer to a buffer containing the caption used for the window.

Video Overlay Extensions

The following additional flags apply to video overlay devices:

MCI_OVLY_WINDOW_DEFAULT
Indicates that the default video window should be used as the target for video.

MCI_OVLY_WINDOW_HWND
This flag indicates the handle of the application window to be used for video. It is included in the *hwndDest* field of the data structure identified by *pParam2* .

MCI_OVLY_WINDOW_STATE
This flag indicates the *usCmdShow* field of the data structure identified by *pParam2* contains a parameter for setting the window state. Window states include:

- SWP_ACTIVATE
- SWP_DEACTIVATE
- SWP_HIDE
- SWP_MAXIMIZE
- SWP_MINIMIZE
- SWP_RESTORE
- SWP_SHOW

Note: The MCI_OVLY_WINDOW_STATE flag only applies to the default window and will *not* affect an application-supplied alternate video window.

MCI_OVLY_WINDOW_TEXT
Indicates that the *pszText* field of the data structure identified by *pParam2* contains a pointer to a buffer containing the caption used for the window.

Note: The MCI_OVLY_WINDOW_TEXT flag only applies to the default window and will *not* affect an application-supplied alternate video window.

pParam2 (PMCI_VID_WINDOW_PARMS)
A pointer to the MCI_VID_WINDOW_PARMS data structure. Devices with additional parameters might replace this pointer with a pointer to a device-specific data structure as follows:

PMCI_DGV_WINDOW_PARMS
A pointer to an MCI_DGV_WINDOW_PARMS data structure.

PMCI_OVLY_WINDOW_PARMS
A pointer to an MCI_OVLY_WINDOW_PARMS data structure.

rc (ULONG)
Return codes indicating success or type of failure:

MCIERR_SUCCESS
MMPM/2 command completed successfully.

MCIERR_OUT_OF_MEMORY
System out of memory.

MCIERR_INVALID_DEVICE_ID
Invalid device ID given.

MCIERR_MISSING_PARAMETER
Missing parameter for this command.

MCIERR_DRIVER
Internal MMPM/2 driver error.

MCIERR_INVALID_FLAG

Invalid flag specified for this command.

MCIERR_MISSING_FLAG
Flag missing for this MMPM/2 command.

MCIERR_FLAGS_NOT_COMPATIBLE
Flags not compatible.

MCIERR_INSTANCE_INACTIVE
Instance inactive.

MCI_WINDOW - Remarks

By default, video devices create a window when an application opens the device, but they do not display it until they receive a **window state show** command or a **play** command. Applications can send the MCI_WINDOW message to tell a video device to use an application window instead of the default window to display video. Applications that supply window handles should be prepared to update an invalid rectangle on the window.

Several flags are provided to allow users to manipulate the window. Because MCI_STATUS can be used to obtain the current window handle, programmers might choose to use the standard window APIs instead. The flags are provided to allow applications that use the string interface to perform standard operations.

Support of this message by a device is optional. The parameters and flags for this message vary according to the selected device.

MCI_WINDOW - Related Messages

- MCI_WHERE

MCI_WINDOW - Example Code

The following code illustrates several examples of how to specify the window and the window characteristics that a graphic device uses with MCI_WINDOW.

```
/* Use for (MCI_DGV_WINDOW_DEFAULT) */
   USHORT  usUserParm = 0;
   ULONG   ulReturn;
   MCI_DGV_WINDOW_PARMS mciWindowParms;

   memset (&mciWindowParms, 0x00, sizeof (MCI_DGV_WINDOW_PARMS));
   mciWindowParms.hwndCallback = hwndNotify;
   mciWindowParms.hwndDest = 0;

   ulReturn = mciSendCommand(usDeviceID, MCI_WINDOW,
                  MCI_WAIT | MCI_DGV_WINDOW_DEFAULT,
                  (PVOID)&mciWindowParms,
                  usUserParm);


   /* Use for MCI_WINDOW (MCI_DGV_WINDOW_HWND) */
   USHORT  usUserParm = 0;
   ULONG   ulReturn;
   MCI_DGV_WINDOW_PARMS mciWindowParms;

   memset (&mciWindowParms, 0x00, sizeof (MCI_DGV_WINDOW_PARMS));
   mciWindowParms.Callback = hwndNotify;
   mciWindowParms.hwndDest = hwndAlternate;
```

```
ulReturn = mciSendCommand(usDeviceID, MCI_WINDOW,
               MCI_WAIT | MCI_DGV_WINDOW_HWND,
               (PVOID)&mciWindowParms,
               usUserParm);

/* Use for MCI_WINDOW (MCI_DGV_WINDOW_STATE) */
USHORT  usUserParm = 0;
ULONG   ulReturn;
MCI_DGV_WINDOW_PARMS mciWindowParms;

/* An example of a message to SHOW the current video window */
memset (&mciWindowParms, 0x00, sizeof (MCI_DGV_WINDOW_PARMS));
mciWindowParms.hwndCallback = hwndNotify;
mciWindowParms.hwndDest = 0;
mciWindowParms.usCmdShow = (INT)SWP_SHOW;

ulReturn = mciSendCommand(usDeviceID, MCI_WINDOW,
               MCI_WAIT | MCI_DGV_WINDOW_STATE,
               (PVOID)&mciWindowParms,
               usUserParm);

/* Use for MCI_WINDOW (MCI_DGV_WINDOW_TEXT) */
USHORT  usUserParm = 0;
ULONG   ulReturn;
MCI_DGV_WINDOW_PARMS mciWindowParms;

memset (&mciWindowParms, 0x00, sizeof (MCI_DGV_WINDOW_PARMS));
mciWindowParms.hwndCallback = hwndNotify;
mciWindowParms.hwndDest = 0;
mciWindowParms.pszText= (PSZ)"New Caption";

ulReturn = mciSendCommand(usDeviceID, MCI_WINDOW,
               MCI_WAIT | MCI_DGV_WINDOW_TEXT,
               (PVOID)&mciWindowParms,
               usUserParm);
```

MCI_WINDOW - Topics

Select an item:
